CANADIAN THESES ON MICROFICHE

I.S.B.N.

THESES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Canada

NL-339 (r. 82/08)

0-315-09018-9

National Library of Canada
Bibliothèque nationale du Canada

133 Canadian Theses Division
Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

56935

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

CAROL JOYCE SMITH

Date of Birth — Date de naissance

10/11/55

Country of Birth — Lieu de naissance

CANADA

Permanent Address — Résidence fixe

10466 28A AVE
EDMONTON AB
T6J 4J6

Title of Thesis — Titre de la thèse

Determining Minimal Planar Partitions for Graphs

University — Université

University of Alberta

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

Master of Science

Year this degree conferred — Année d'obtention de ce grade

1982

Name of Supervisor — Nom du directeur de thèse

Dr. Wayne Jackson

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date

Jan 14/82

Signature

Carol Smith

THE UNIVERSITY OF ALBERTA

Determining Minimal Planar Partitions for Graphs

by

© Carol Joyce Smith

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF Master of Science

Computing Science

EDMONTON, ALBERTA

SPRING, 1982

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR Carol Joyce Smith

TITLE OF THESIS Determining Minimal Planar Partitions for Graphs

DEGREE FOR WHICH THESIS WAS PRESENTED Master of Science

YEAR THIS DEGREE GRANTED 1982

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(SIGNED) Carol Smith

PERMANENT ADDRESS:

10466 28A Avenue,

Edmonton, Alberta

T6J 4J6

DATED November 24, 1981

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled

Determining Minimal Planar Partitions for Graphs

submitted by Carol Joyce Smith in partial fulfilment of the requirements for the degree of Master of Science.

L W Jacobs
Supervisor

J W Moon

Date: November 24, 1981

Abstract

Minimal planar partitions have been determined for the class of complete graphs, most complete bipartite graphs, and the class of m-cubes. In addition, minimal planar partitions whose subgraphs have vertex degrees of at most four have been determined for most complete graphs and all complete bipartite graphs. However, there does not exist at present a method for determining a minimal planar partition for an arbitrary graph in polynomial time. Edge partitioning algorithms are one means of obtaining partial (approximate) solutions to this problem.

The purpose of this thesis is to examine the results obtained so far on minimal planar partitions for the classes of graphs mentioned above, and to introduce three efficient planar edge partitioning algorithms that produce good approximations (in terms of size) to a minimal planar partition for an arbitrary graph and achieve a balance between their complexity and their performance.

Acknowledgments

I wish to thank my supervisor Dr Wayne Jackson for his help and encouragement and Lisa Higham and Dr Anne Brindle for their persistence in reading the drafts of this thesis and their many helpful suggestions. My thanks also to Dr D. J. Miller for introducing me to the areas of graph planarity and graph thickness. I also wish to thank Sandra Taylor for her work on the figures in this thesis.

Finally, my deepest appreciation to Ken who, in addition to reading this thesis, put up with the long nights and weekends spent on this work.

Table of Contents

Chapter Page

1. Introduction 1

2. History of the Minimal Planar Partition Problem 5

   2.1 Graph Thickness 8

      2.1.1 Upper and Lower Bounds 8

      2.1.2 Complete Graphs, Complete Bipartite Graphs, and M-Cubes 9

      2.1.3 Complete Graphs and Complete Bipartite Graphs with Degree Constrained Partitions ...12

   2.2 The Construction of Minimal Planar Partitions 16

      2.2.1 Partitioning Complete Graphs 17

      2.2.2 Partitioning Complete Bipartite Graphs 20

      2.2.3 Partitioning Complete Graphs and Complete Bipartite Graphs Into Degree Constrained Partitions 24

3. Planarity Algorithms 32

   3.1 A Path Finding Planarity Algorithm 32

      3.1.1 A Special Ordering of the Adjacency Lists of G 33

      3.1.2 The Embedding Procedure 35

      3.1.3 The Formation and Use of Blocks 50

      3.1.4 A Correction to the Implementation of the Embedding Procedure 57

   3.2 A pq-tree Planarity Algorithm 58

      3.2.1 An st-Numbering of G 59

      3.2.2 The pq-tree Embedding Procedure 64

4. Edge Partitioning Algorithms 80

   4.1 The Spanning Tree Edge Partitioning Algorithm 83

      4.1.1 The Correctness of Algorithm 4.1.1 (STEP) ...84

4.1.2 Analysis of Time Complexity 85
4.1.3 The STEP Algorithm Summary 85
4.2 The Path Finding Edge Partitioning Algorithm 86
4.2.1 The Planar Graph Constructor 87
4.2.2 The Correctness of Algorithm 4.2.1 (PFEP) ...92
4.2.3 Analysis of Time Complexity 99
4.2.4 The PFEP Algorithm Summary 99
4.3 The pq-tree Edge Partition Algorithm 101
4.3.1 The Planar Graph Constructor 102
4.3.2 The Correctness of Algorithm 4.3.1 (PQEP) ..117
4.3.3 Analysis of Time Complexity 119
4.3.4 The PQEP Algorithm Summary 122
5. Experimental Analysis of the STEP, PFEP, and PQEP Algorithms 124
5.1 The Experiments 124
5.2 Performance Measure 125
5.3 Performance of the STEP, PFEP, and PQEP Algorithms 126
5.4 Resource Requirements 130
5.5 Characteristics of the Constructed Planar Partitions 133
6. Summary and Conclusions 147
6.1 Summary 147
6.2 Conclusions 150
6.3 Further Research 152
References 154

List of Tables

Table		Page
I	Test Graphs	126
II	The Number of Subgraphs t(G) in a Minimal Planar Partition of G and the Number of Subgraphs in the Constructed Planar Partitions	127
III	Performance Measure PM of the STEP, PQEP, and PFEP Algorithms	130
IV	Execution Time in Seconds	131
V	Number of Edges in the Largest Planar Subgraph	134

List of Figures

Figure		Page
1.	A Planar Partition of K(6,7)	16
2.	Hr and Fh	19
3.	The subgraph H1 of K18	20
4.	Partitioning K(m,p)	23
5.	Degree Constrained Partitioning of Kn	26
6.	Degree Constrained Partitioning of K(4r+2,4r+2)	29
7.	Degree Constrained Partitioning of K(m,4r+4)	31
8.	Reachable Vertices	35
9.	The Decomposition of a Graph	37
10.	S and S' Interlace	39
11.	S U C	44
12.	S is embedded before S'	46
13.	S and S' interlace if and only if f < w'	48
14.	An st-numbering of a Graph	64
15.	Real and Virtual Vertices	66
16.	Embeddings and the Corresponding pq-trees	69

17. Leaf-nodes that can and cannot be made adjacent 72

18. Blocked and Unblocked Nodes 74

19. The Form of T_k after a bubble pass 75

20. Transforming T_3 77

21. Deleting a Back Edge 89

22. Loss of Interaction 99

23. T_k nonplanar after a bubble pass 104

24. The Form of q-nodes in a pq-tree 105

25. The Form of an Unmatched p-node 106

26. A Possible Configuration for an Unmatched q-node ...108

27. Possible Configurations of the Children of x_i 108

28. Matchable Pertinent Groups of a q-node 110

29. T_k before and after deletion of leaf-nodes during the Reduction Pass 113

30. Embedding a Graph whose Vertices are not st-Numbered 120

31. Subgraphs Constructed by PFEP 137

32. Subgraphs Constructed by PQEP 141

33. Subgraphs Constructed by STEP 144

Chapter 1

Introduction

This thesis examines the problem of determining a minimal planar partition for a graph. The problem involves partitioning the edges of a graph into the smallest number of subgraphs such that each subgraph can be embedded in the plane with no edges crossing. The minimal planar partition problem or one closely related to it often occurs in designing the layout of an integrated circuit [Fer 70, Rub 73, Han 75, Van 75, Lui 76]. The circuit is represented as a graph, where the components and wires of the circuit are the vertices and edges of the graph respectively. The goal is to place the wires of the circuit on the minimum number of boards such that wires on the same board do not cross.

At present, the minimal planar partition problem is not solvable in polynomial time for an arbitrary graph. In addition, it is not known whether the problem is NP-complete [Gar 79]. However, it is known that the related problem of nonplanar edge deletion is NP-complete [Lui 76]. The nonplanar edge deletion problem involves determining the minimum number of edges whose deletion from a graph results in a planar subgraph.

It is reasonable to assume that the minimal planar partition problem is not in P and to look for algorithms that obtain partial solutions to the problem in polynomial

time rather than an exact solution.

This thesis has two goals: (1) to design an efficient algorithm that produces a good approximate solution to instances of the minimal planar partition problem, and (2) to achieve a balance between the complexity of the algorithm and the quality of the approximation it produces, that is, to find the point where any improvement in the performance of the algorithm requires a disproportionate increase in its complexity.

Theoretical results on the minimal planar partition problem are surveyed in Chapter 2. The results surveyed deal with the order of a minimal planar partition for a graph (the thickness of a graph) and the construction of some minimal planar partitions. Included are results from Tutte [Tut1 63], Beineke and Harary [Bei 65 and Bei1 67], Beineke, Harary, and Moon [Bei 64 and Bei2 67], Alekseev and Goncakov [Ale 76], Kleinert [Hob 69], and Bose and Prahbu [Bos 77]. The main result from Tutte is an upper bound on the thickness of a graph. The papers by Beineke and Harary and by Alekseev and Goncakov are concerned with minimal planar partitions of complete graphs. The results from Beineke, Harary, and Moon deal with minimal planar partitions of complete bipartite graphs and Kleinert's result [Hob 69] concerns the thicknesses of m-cubes.

Also included in the survey are the results from Bose and Prahbu on minimal planar partitions of complete graphs and complete bipartite graphs where the degree of a vertex

in any subgraph of the partition is restricted to be less than or equal to some integer d. In particular, their solution for the case d = 4 is examined in section 2.1.3 and expanded on.

Chapter 3 is devoted to the examination of two linear time planarity algorithms: that of Hopcroft and Tarjan, and that of Booth and Lueker. The path finding algorithm from Hopcroft and Tarjan [Hop 74] contains several minor errors which have been noted in the literature [Deo 76] and one major, subtle logic error which is noted in section 3.1.4. The pq-tree algorithm from Booth and Lueker [Boo 76] is an O(n) version of an earlier vertex embedding algorithm suggested by Lempel, Even, and Cederbaum [Lem 66]. These two algorithms are used in developing the planar edge partitioning algorithms described in Chapter 4.

Chapter 4 examines three new edge partitioning algorithms that produce a planar partition of a graph. Two of the algorithms proposed are based on the algorithms in Chapter 3. The first proposed algorithm is a spanning tree edge partitioning (STEP) algorithm. To produce a planar subgraph in a partition this algorithm starts with a spanning tree over the vertices of G and the edges of G not yet partitioned. Edges are added one at a time to this initial subgraph while maintaining its planarity. The second algorithm, a path finding edge partitioning (PFEP) algorithm, uses a modified version of the path finding planarity algorithm of Chapter 3 to partition the edges of G

into a set of planar subgraphs. The third proposed algorithm, a pq-tree edge partitioning (PQEP) algorithm, has the same overall structure as the PFEP algorithm but makes use of a modified version of the pq-tree planarity algorithm of Chapter 3. Theoretical results for the new algorithms include a proof of the correctness of each algorithm and an upper bound on the time complexity of each algorithm.

The value of the three new algorithms as heuristic algorithms for the minimal planar partition problem is assessed in Chapter 5. One performance measure is considered and experimental results from tests run on all three algorithms are presented and compared. The algorithms are also compared on the basis of their resource requirements (execution time) and the characteristics of the constructed planar partitions (the number of edges in the largest subgraph and the properties of each subgraph).

Chapter 6 summarizes the three new algorithms and their experimental behavior, discusses the merits and drawbacks of each, relates the goals of this thesis to what was accomplished, and presents ideas for further research.

Chapter 2

History of the Minimal Planar Partition Problem

This chapter surveys theoretical results on the minimal planar partition problem. Results on the order of the minimal planar partition for a graph (the thickness of a graph) are presented first. Included in these results are upper and lower bounds on a graph's thickness and the thicknesses of the complete graphs, the complete bipartite graphs, and the m-cubes. Also examined are results on the minimum size of planar partitions of complete graphs and complete bipartite graphs where each subgraph in the partition has vertices of degree less than or equal to four (degree constrained partitions).

Then the methods used in constructing minimal planar partitions and minimum size, degree constrained planar partitions of some complete graphs and some complete bipartite graphs are examined.

Let G = (V, E) represent a simple, undirected graph where V is the set of vertices of G and E is the set of edges of G. Let n be the number of vertices in V and let e be the number of edges in E. Where necessary, the notation V(G) and E(G) is used for V and E respectively to avoid ambiguity.

Definition 2.1

A graph G is **connected** if every pair of vertices is joined by a path. A maximal connected subgraph of G is called a **connected component** of G.

Definition 2.2

A graph G is **biconnected** if every pair of vertices lies on a cycle. A maximal biconnected subgraph of G is called a **biconnected component** of G.

Definition 2.3

A **planar embedding** of a graph G is an embedding in a plane (or on a surface of a sphere) of the vertices and edges of G such that each vertex of G is at a different location in the plane and two edges intersect only at their common vertex if they have a vertex in common.

Definition 2.4

The regions defined by a planar embedding of a graph are the **faces** of the planar embedding.

Definition 2.5

A graph G is **planar** if it has at least one planar embedding, otherwise, G is **nonplanar**.

Definition 2.6 [Tut1 63]

A **partition** of order k of a graph G is the decomposition of G into k subgraphs Hi, $1 \leq i \leq k$, such that each Hi has at least one edge and any two subgraphs are edge

disjoint but not necessarily vertex disjoint.

Definition 2.7 [Tut1 63]

A **partition** is **planar** if each subgraph of the partition is planar.

Definition 2.8 [Tut1 63]

The **thickness** t(G) of a graph G is k if G possesses a planar partition of order k and G does not possess a planar partition of order less than k.

The term **minimal planar partition** of G is used to refer to a planar partition of G with order t(G). Note that G may have more than one minimal planar partition.

The complete graph is denoted by Kn and is the graph with n vertices in which every pair of vertices is joined by an edge. A graph G is bipartite if its set V of vertices can be divided into two nonempty, disjoint subsets V_1 and V_2 such that every edge in E connects a vertex in V_1 to a vertex in V_2. Let $m = |V_1|$ and let $p = |V_2|$. If each vertex in V_1 is joined to every vertex in V_2, then G is a complete bipartite graph and is denoted by K(m,p). The m-cube [Har 69 and Bon 76], denoted by Qm, is a graph with 2**m vertices and m2**(m-1) edges. The vertices of Qm are numbered from 0 to 2**m - 1 with the numbers represented in their m-digit binary form. An edge joins two vertices if and only if their binary representations differ in exactly one digit.

Let [x] be the greatest integer less than or equal to a real number x and let {x} be the smallest integer greater than or equal to x.

2.1 Graph Thickness

2.1.1 Upper and Lower Bounds

The results surveyed in this section pertain to upper and lower bounds on the thickness of a graph. The lower bound of Theorem 2.1.2 and other results mentioned later are obtained using Euler's formula.

Theorem 2.1.1 (Euler's Formula) A connected planar graph with n vertices, e edges, and f faces satisfies

$$n - e + f = 2.$$

Theorem 2.1.2 The thickness t(G) of a graph with n vertices and e edges satisfies

$$t(G) \geq e/(3n-6).$$

Proof: In a planar embedding of a graph, every face is composed of at least three edges and every edge borders on two faces. Therefore, $3f \leq 2e$ or $f \leq 2e/3$. For a planar graph

$$n - 2 = e - f \geq e - 2e/3,$$

which simplifies to

$$e \leq 3n - 6.$$

Therefore, any planar subgraph of G has at most 3n - 6

edges and $t(G) \geq e/(3n - 6)$.

Theorem 2.1.3 (Tutte [Tut1 63]) Let G be a graph of thickness t(G). Let G' be formed from G by deleting either a single edge or a single vertex with all its incident edges. Then the thickness of G' is either t(G) or t(G)-1.

Proof: Let the thickness of G' be k. Since G' is a subgraph of G, $k \leq t(G)$. Since a planar partition for G of order k+1 can be formed by adding one subgraph of the deleted elements to G', $t(G) \leq k+1$. Thus $t(G)-1 \leq k \leq t(G)$.

Corollary 2.1.4 Let G be any graph with $m \leq n$ vertices, then $t(G) \leq t(Kn)$.

2.1.2 Complete Graphs, Complete Bipartite Graphs, and M-Cubes

Interest in the thicknesses of complete graphs was kindled about 1962 when Battle, Harary, and Kodama [Bat 62] and Tutte [Tut2 63] proved that t(K9) = 3. The thicknesses of five out of every six complete graphs were established in 1965 by Beineke and Harary [Bei 65 and Bei1 67]. Their work was extended by Alekseev and Goncakov [Ale 76] (and independently by Vasak [Vas 76]) to include all complete graphs.

Theorem 2.1.5 For a complete graph Kn,

$$t(Kn) \geq [(n+7)/6].$$

Proof: $t(Kn) \geq e/(3n - 6)$ by Theorem 2.1.2.

Since the number of edges e in Kn is $n(n-1)/2$,

$$t(Kn) \geq n(n - 1)/(6n - 12)$$

$$\geq [(n + 7)/6].$$

Theorem 2.1.6 [Bei 65 and Bei1 67] For $n \neq 4(\text{mod } 6)$, and $n \neq 9$,

$$t(Kn) = [(n+7)/6].$$

Theorem 2.1.7 [Ale 76] The thickness of Kn for $n \neq 9, 10$ is

$$t(Kn) = [(n+7)/6],$$

and

$$t(K9) = t(K10) = 3.$$

The proof of Theorem 2.1.6 involves using Theorem 2.1.5 and constructing a planar partition of order $[(n+7)/6]$ for Kn, when $n \neq 4(\text{mod } 6)$ and $n \neq 9$. The construction of the planar partition is explained in detail in section 2.2.1. By modifying the construction technique from Beineke [Bei1 67] slightly, Alekseev and Goncakov [Ale 76] were able to construct a planar partition of order $[(n+7)/6]$ for graphs Kn, $n = 4(\text{mod } 6)$, $n \geq 22$, thereby establishing Theorem 2.1.7.

Thickness results for the bipartite graphs first appeared in a paper by Beineke, Harary, and Moon [Bei 64] and then in more detail in a paper by Beineke [Bei2 67]. The

main results are illustrated by the following four theorems.

Theorem 2.1.8 If G is bipartite, then

$$t(G) \geq \{e/(2n - 4)\}.$$

Proof: For a bipartite graph G each face has at least four edges and therefore, by Theorem 2.1.1, any planar subgraph of G will have at most 2n - 4 edges. It follows then that

$$t(G) \geq \{e/(2n - 4)\}.$$

Theorem 2.1.9 The thickness of K(m,p) satisfies

$\{mp/(2m+2p-4)\} \leq t(K(m,p)) \leq \{m/2\}$, for $2 < m \leq p$.

Proof: The graphs K(1,p) and K(2,p) are planar and K(m,p) is isomorphic to K(p,m) so we only have to consider complete bipartite graphs for which $2 < m \leq p$.

1. $t(K(m,p)) \geq \{mp/(2m+2p-4)\}$ follows from Theorem 2.1.8.
2. The proof that $t(K(m,p)) \leq \{m/2\}$ is accomplished in two parts. First, a planar partition of order s is constructed for K(2s,p) by making s copies of K(2,p), thereby proving the conjecture for even values of m. Then, since $t(K(2s-1,p)) \leq t(K(2s,p))$ by Theorem 2.1.3, it follows that $t(K(m,p)) \leq \{m/2\}$ for all values of m.

Theorem 2.1.10 If m is even and $p > (m-2)^2/2$, then

$$t(K(m,p)) = m/2.$$

If m is odd and $p > (m-1)(m-2)$, then

$$t(K(m,p)) = (m+1)/2.$$

Theorem 2.1.11 The thickness of K(m,p), for $m \leq p$ is

$$t(K(m,p)) = \{mp/(2m+2p-4)\},$$

except possibly when m and p are both odd and there exists an integer k such that

$$p = [2k(m-2)/(m-2k)].$$

The proof of Theorem 2.1.11 involves using Theorem 2.1.9 and constructing a planar partition of order $\{mp/(2m+2p-4)\}$ for K(m,p) with m and p as given in the statement of the theorem. The construction of a planar partition for K(m,p) is described in section 2.2.2.

A final result on the thicknesses of complete bipartite graphs, which is stated by Beineke without proof in [Bei2 67], is that

$$t(K(4r-3,4r+1)) = r, \text{ for } r \geq 4.$$

The final result on graph thickness that is surveyed here is the result on the thicknesses of m-cubes as determined by Kleinert [Hob 69].

Theorem 2.1.12 The thickness of an m-cube Qm is

$$t(Qm) = 1 + [m/4].$$

2.1.3 Complete Graphs and Complete Bipartite Graphs with Degree Constrained Partitions

Bose and Prabhu [Bos 77] studied the minimum sizes of planar partitions of graphs (thicknesses of graphs) where the degree of each vertex d(v) in the subgraphs of the planar partition is constrained to be less than or equal to

some value d. In particular, the problem is studied for the case d = 4 and is solved for most complete graphs and complete bipartite graphs. It is conjectured in their paper that the thickness of the complete bipartite graph, K(m,p), for $m = 4r+2$, $p = 4r+3$, $r \geq 1$, is

$$t(K(m,p)) = [(m+5)/4]+1.$$

It is shown in this section that this conjecture is false and that, for $m = 4r+2$, $p = 4r+3$, $r \geq 1$,

$$t(K(m,p)) = [(m+5)/4].$$

With the above result, the degree constrained thickness problem is solved for all complete bipartite graphs.

Theorem 2.1.13 Suppose $n > 5$ and $n \neq 4p+1$, where p is an integer and $p \geq 3$. Then the thickness of the complete graph Kn, in the degree constrained case, where d = 4, is given by

$$t(Kn) = [(n+3)/4].$$

The proof of this theorem is accomplished in part by the construction of a planar partition of order $[(n+3)/4]$ for Kn where each subgraph in the planar partition has vertices with degree $d(v) \leq 4$. An outline of this construction is given in section 2.2.3.

Theorem 2.1.14 The thickness of the complete bipartite graph K(m,p), in the degree constrained case, where d = 4 and $m \leq p$, is:

(a) $t(K(m,p)) = [(m+5)/4]$, if $m = p$

(b) $t(K(m,p)) = [(p+3)/4]$, if $m \leq p-2$

(c) $t(K(m,p)) = [(m+5)/4]$, if $m = p-1$

Proof:

(a) $t(K(m,p)) = [(m+5)/4]$, if $m = p$.

First a planar partition of order $r+1$ is constructed for $K(4r+2,4r+2)$, for any positive integer r, establishing

$$t(K(4r+2,4r+2)) \leq r+1. \qquad (1)$$

This construction is shown in more detail in section 2.2.3. Next, since $\{x/y\} = [(x+y-1)/y]$,

$$\begin{aligned} t(K(m,m)) &\geq \{m^2/(4m-4)\} \\ &= [(m^2+4m-4-1)/(4m-4)] \\ &= [(m+5)/4]. \end{aligned} \qquad (2)$$

By (1) and (2), $t(K(4r+2,4r+2)) = r+1$.

Since $t(K(4r-1,4r-1)) \geq r+1$ by (2) and $t(K(4r-1,4r-1)) \leq t(K(4r,4r)) \leq t(K(4r+1,4r+1)) \leq t(K(4r+2,4r+2)) = r+1$, it follows that

$t(K(m,m)) = [(m+5)/4]$ for all m.

(b) $t(K(m,p)) = [(p+3)/4]$, if $m \leq p-2$.

The proof involves the construction of a planar partition of order $r+1$, for $p = 4r+4$, r a positive integer, and $m \leq 4r+2$. The construction is outlined in section 2.2.3 and establishes that

$$t(K(m,4r+4)) \leq r+1.$$

Since $t(K(m,4r+4)) \geq \{m(4r+4)/(4m)\} = r+1$ (a degree constrained planar subgraph can contain $e \leq \min(4m, 2m+2p-4)$ edges), it follows that $t(K(m,4r+4)) = r+1$. In addition, since

$t(K(m,4r+1)) \geq \{m(4r+1)/(4m)\} = r+1$ and

$t(K(m,4r+1)) \leq t(K(m,4r+2)) \leq t(K(m,4r+3)) \leq$

$t(K(m,4r+4))$, it follows that

$$t(K(m,p)) = [(p+3)/4], \text{ for } m \leq p-2.$$

(c) $t(K(m,p)) = [(m+5)/4]$, if $m = p-1$.

Proof: $t(K(m,m)) \leq t(K(m,m+1)) \leq t(K(m+1,m+1))$ or,

$[(m+5)/4] \leq t(K(m,m+1)) \leq [(m+6)/4]$ by part (a). (3)

i) For $m = 4r-1$, $r \geq 1$,

$t(K(4r-1,4r)) = r+1$ from (3),

ii) For $m = 4r$, $r \geq 1$,

$t(K(4r,4r+1)) = r+1$ from (3),

iii) For $m = 4r+1$, $r \geq 1$,

$t(K(4r+1,4r+2)) = r+1$ from (3), and

iv) For $m = 4r+2$, $r \geq 1$,

$t(K(4r+2,4r+3)) = r+1$.

This is established as follows:

1. $r+1 \leq t(K(4r+2,4r+3)) \leq r+2$ by (3).
2. Although it is conjectured that $t(K(4r+2,4r+3)) = r+2$ [Bos 77], it is easy to show that the conjecture is false. For example, if $r = 1$, then $K(4r+2,4r+3) = K(6,7)$ and $t(K(6,7)) = 2$, not 3, as shown in Figure 1. Now, from part (a),

$$t(K(4r+2,4r+2)) = r+1, \text{ and,}$$

from part (b),

$$t(K(4r+2,4r+4)) = r+1.$$

Therefore, $r+1 = t(K(4r+2,4r+2)) \leq$

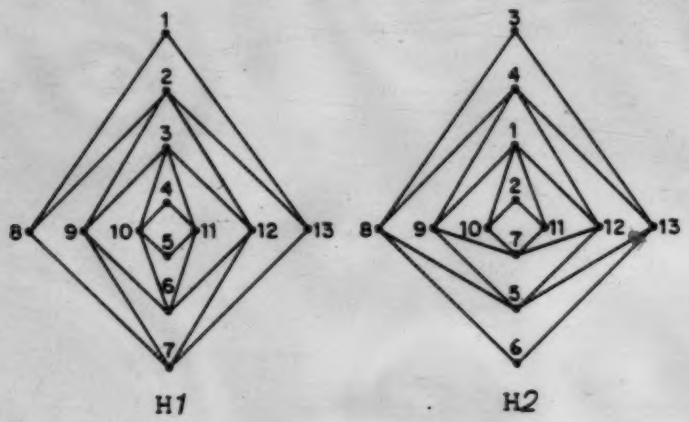


Figure 1. A Planar Partition of K(6,7)

$t(K(4r+2,4r+3)) \leq t(K(4r+2,4r+4)) = r+1$, and hence

$$t(K(4r+2,4r+3)) = r+1.$$

From i, ii, iii, and iv above, it follows that

$$t(K(m,m+1)) = [(m+5)/4], \text{ for all } m.$$

2.2 The Construction of Minimal Planar Partitions

The methods used to construct minimal planar partitions and minimum size, degree constrained planar partitions of complete graphs and complete bipartite graphs are examined in this section. There are two reasons for this examination. The first reason is to unify the construction methods used to produce the different partitions. The second reason is to indicate the structure of the partitions and to indicate what is involved in determining a partition.

2.2.1 Partitioning Complete Graphs

Beineke and Harary [Bei 65] and Beineke [Bei1 67] construct planar partitions for some complete graphs as part of the proof of Theorem 2.1.6. This section examines the construction of a planar partition, of order k+1, for a complete graph Kn, where n = 6k, k ≥ 1. Let the vertices of Kn be numbered from 1 to n. The construction technique shown here is the one described in [Bei1 67]. The planar partition constructed for Kn is obtained from a k by k array A defined as:

$$A(i,j) = \{((-1)^{**}i)*[i/2] + ((-1)^{**}j)*[j/2]\} \pmod{k},$$
$$\text{for } 1 \le i,j \le k.$$

Every integer between 1 and k appears exactly once in every row and column of A.

Example 2.2.1 For k = 3, n = 18,

$$A = \begin{matrix} 3 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 1 \end{matrix}$$

The construction of the planar partition for Kn also requires the following function q. For any two integers r and s, r,s ≤ k, let j be the column of A where A(1,j) = r, and let i be the row in column j where A(i,j) = s. Then,

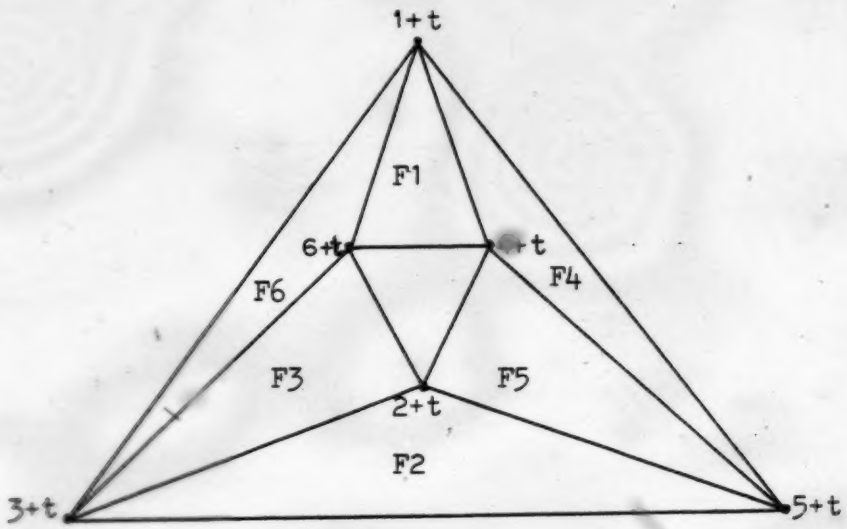
$$q(r,s) = (-1)^{**}(1+\min(i,j)).$$

The subgraphs H_r of the planar partition for Kn, r = 1, 2, ..., k, are constructed as follows. Each subgraph H_r has the n vertices of Kn. The general form of H_r is illustrated

in Figure 2(a). Each of the subgraphs F_h of H_r, $h = 1, 2, \ldots, 6$, has the general form shown in Figure 2(b). F_h has $k-1$ interior vertices, three exterior vertices, $3k-3$ interior edges, and three exterior edges. The exterior vertices and edges of F_h are the vertices and edges in H_r that define the face of H_r in which F_h lies (as illustrated in Figure 2(a)). The labels on the $k-1$ interior vertices of each F_h in H_r are determined using the function q and the entries in the column j of A for which $A(1,j) = r$.
Let $s = A(i,j)$, then the label on the $(i-1)$'st interior vertex of F_h, $i = 2, 3, \ldots, k$, $h = 1, 2, \ldots, 6$, is

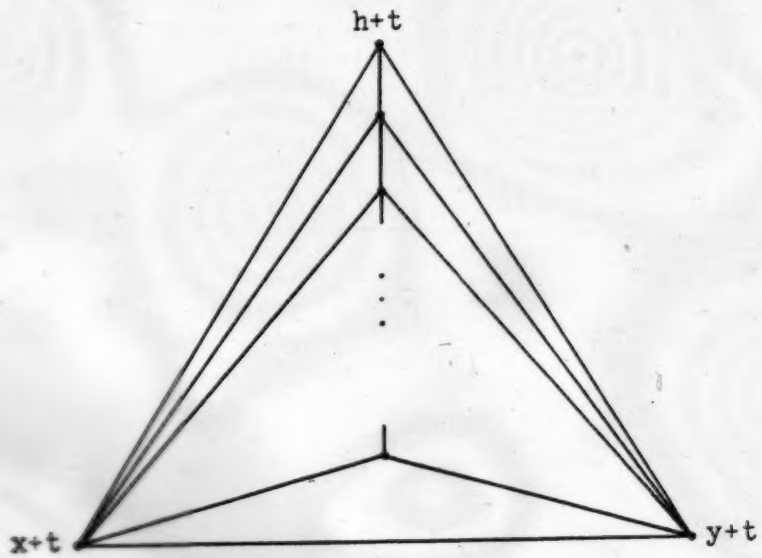
1. $h+6(s-1)$, if $q(r,s) = +1$,
2. $(h+1)+6(s-1)$, if $q(r,s) = -1$ and h is odd,
3. $(h-1)+6(s-1)$, if $q(r,s) = -1$, and h is even.

Another subgraph H_{k+1} is necessary to complete the planar partition of Kn. The subgraph H_{k+1} contains the n vertices of Kn and the 3k edges $(1+t,2+t)$, $(3+t,4+t)$, $(5+t,6+t)$, for $t = 6(r-1)$, $r = 1, 2, \ldots, k$. The set $(H_1, H_2, \ldots, H_{k+1})$ is a planar partition of Kn, for $n = 6k$. It is minimal because $t(K_{6k}) \geq k+1$ by Theorem 2.1.5. Figure 3 shows a planar embedding of the subgraph H_1 of K_{18} constructed using the array A from example 2.2.1.



t = 6(r-1)

(a) The General Form of Hr



x = (h-1)(mod 6), if h is odd
(h+1)(mod 6), otherwise
y = (h+3)(mod 6)

(b) The General Form of Fh

Figure 2. Hr and Fh

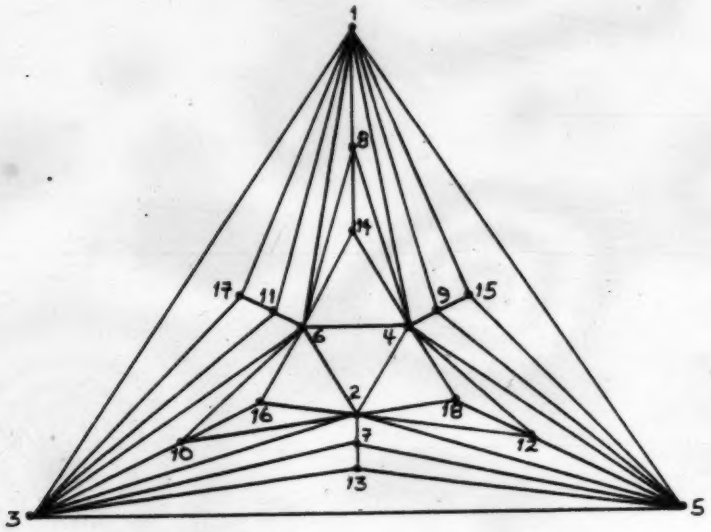


Figure 3. The subgraph H7 of K18

2.2.2 Partitioning Complete Bipartite Graphs

The proof of Theorem 2.1.11 depends on the construction of planar partitions for certain complete bipartite graphs K(m,p). The construction outlined in this section appears in the paper by Beineke [Bei1 67]. Let k and m be integers satisfying $k < m/2$. Let g be a function defined by

$$g(m,k) = [2k(m-2)/(m-2k)].$$

If p is chosen as the largest even integer not exceeding g(m,k), then

$$\{mp/(2m+2p-4)\} = k.$$

Therefore, if a planar partition of order k is constructed for K(m,p), it follows that

$$t(K(m,p)) \leq \{mp/(2m+2p-4)\}$$

for certain values of m and p.

With respect to the construction of a partition of order k, let s be an integer, s = p/2 (p even), and let the vertices of K(m,p) be numbered from 1 to m+p with the vertices of the first vertex set numbered from 1 to m, and the vertices of the second vertex set numbered from m+1 to m+p.

To construct a planar partition for K(m,p), an s by k array A is used. The elements A(i,j) are variable length vectors. To form A, first an s by k array C is constructed, where

$$C(i,j) = \{(i+j)m/k\} - \{(i+j-1)m/k\}.$$

C(i,j) is the length of the element A(i,j). The element A(1,1) is

$$A(1,1) = (1, 2, \ldots, C(1,1)).$$

The remaining elements of A are defined inductively. For any row i of A, $1 \le i \le s$, the entries of element A(i,j), $1 \le j \le k$, are consecutive integers (modulo m). The first entry of A(i,j) is consecutive (modulo m) with the last entry of A(i,j-1). For any column j of A, $1 \le j \le k$, the first entry of A(i,j) is equal to the second-to-last entry of A(i-1,j), $2 \le i \le k$.

Example 2.2.2 Let m=8 and k=3, then p=18 and

$$C = \begin{matrix} 3 & 2 & 3 \\ 2 & 3 & 3 \\ 3 & 3 & 2 \\ 3 & 2 & 3 \end{matrix}$$

2 3 3

3 3 2

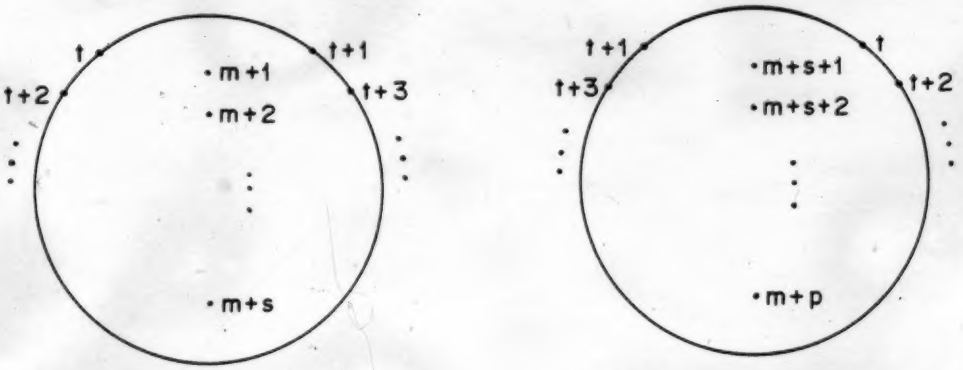
3 2 3

2 3 3

3 3 2

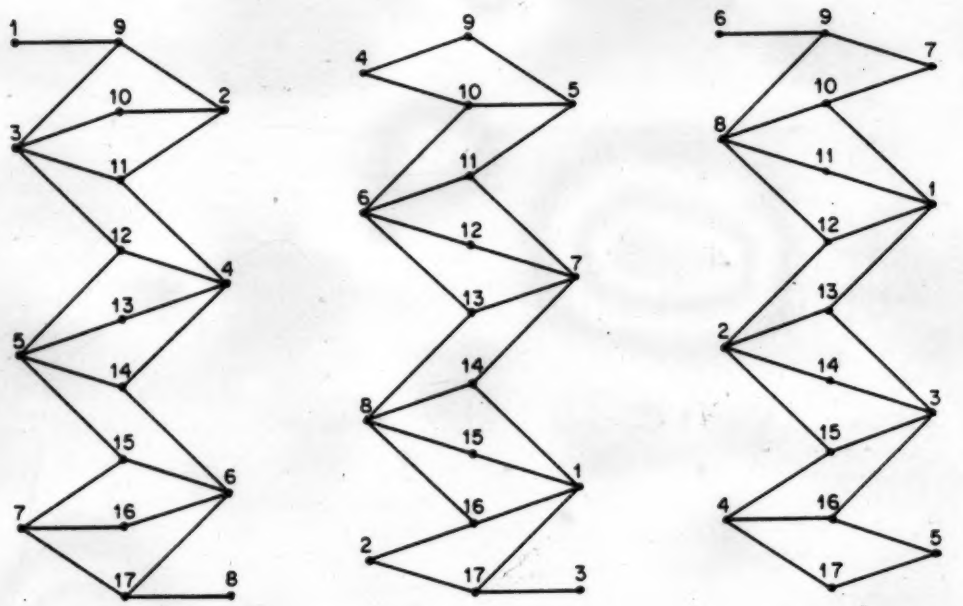
A =			
	1 2 3	4 5	6 7 8
	2 3	4 5 6	7 8 1
	2 3 4	5 6 7	8 1
	3 4 5	6 7	8 1 2
	4 5	6 7 8	1 2 3
	4 5 6	7 8 1	2 3
	5 6 7	8 1	2 3 4
	6 7	8 1 2	3 4 5
	6 7 8	1 2 3	4 5

The entries of each element A(i,j) are integers between 1 and m, and represent the vertices in the first vertex set of K(m,p). The row numbers 1, 2, ..., s of A represent the vertices in the second vertex set of K(m,p) where row i represents vertices m+i and m+s+i. For j = 1, 2 ,..., k, each subgraph H*j* has the following planar embedding. The vertices of H*j* are the vertices of K(m,p) and are placed on a sphere as depicted in Figure 4(a), where t is the first entry of A(1,j). Each column j of A, $1 \leq j \leq k$, specifies which edges of K(m,p) are in H*j*. The vertices represented in A(i,j), $1 \leq i \leq s$, (from the first vertex set) are adjacent



(i) Upper Hemisphere (ii) Lower Hemisphere

(a) The Position of the Vertices on the Sphere



(iii) H1 (iv) H2 (v) H3

(b) The Upper Hemisphere of each subgraph for K(8,18)

Figure 4. Partitioning K(m,p)

to the vertices m+i and m+i+s (from the second vertex set) in H_j. Each edge of H_j is drawn on the surface of the sphere using the line of shortest length between its two endpoints. Figure 4(b) shows the upper hemisphere of each subgraph H_j for K(8,18), $1 \leq j \leq 3$. The subgraphs are constructed using the array A defined in example 2.2.2.

The array A has the following properties:

1. Each integer between 1 and m appears once and only once in each row of A. This means that each vertex 1, 2, ..., m is connected to each vertex m+1, m+2, ..., m+p once and only once. Thus, A represents all the edges of K(m,p).
2. Only the last two entries of any element A(i,j) appear in elements below A(i,j) in column j. This property ensures that the subgraph whose edges are specified by column j is planar.

2.2.3 Partitioning Complete Graphs and Complete Bipartite Graphs Into Degree Constrained Partitions

The proof of Theorem 2.1.13 is accomplished in part by the construction of a degree constrained planar partition of order p for a complete graph Kn, where n = 4p, and p is a positive integer. The subgraphs H_k in the partition, $1 \leq k \leq p$, have vertices of degree $d(v) \leq 4$. Although Bose and Prabhu do not construct the planar subgraphs using an array it is done here to achieve consistency.

For n = 4p, a 2p by 2p array A is defined as:

$A(1,j) = j,$

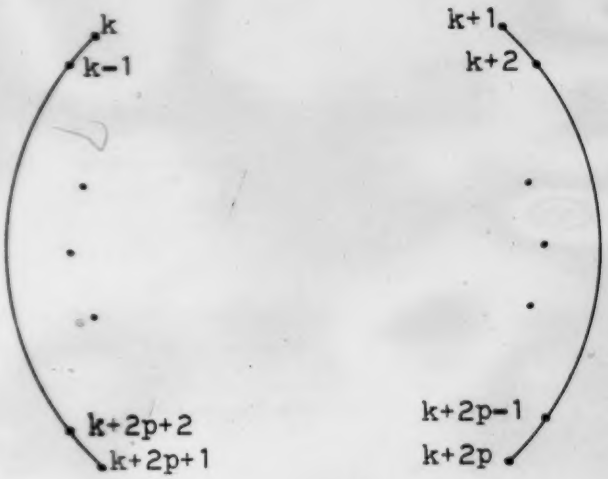
$A(i,j) = (j-i+1,\ j-i+2) \pmod{4p},$

for $1 \le j \le 2p$, and $2 \le i \le 2p$.

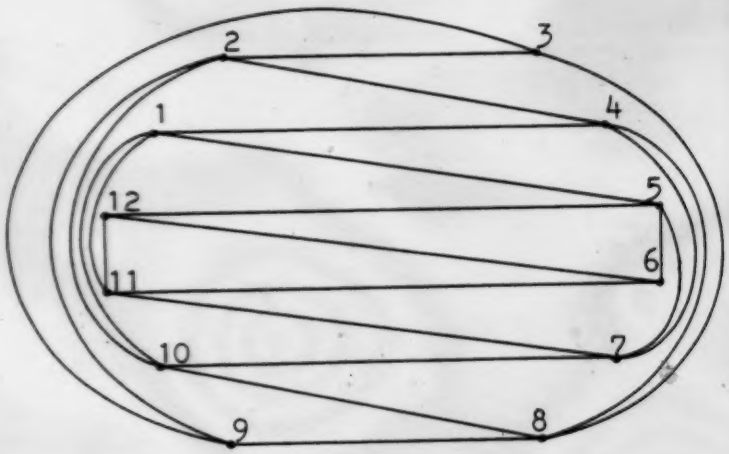
Example 2.2.3 For n=12, p=3, A is a 6 by 6 array

A =	1	2	3	4	5	6
	12 1	1 2	2 3	3 4	4 5	5 6
	11 12	12 1	1 2	2 3	3 4	4 5
	10 11	11 12	12 1	1 2	2 3	3 4
	9 10	10 11	11 12	12 1	1 2	2 3
	8 9	9 10	10 11	11 12	12 1	1 2

Let the vertices of Kn be numbered from 1 to n. Each subgraph H*k*, k = 1, 2, ..., p, has the same vertices as Kn. In the planar embedding of H*k*, the vertices are arranged on the sphere as illustrated in Figure 5(a). Both column k and column 2p+1-k of A are used in the construction of H*k*. The only edges in H*k* are those edges between vertex k+i and each entry in A(i,k), and those edges between vertex (2p+1-k)+i and each entry of A(i,2p+1-k) for i = 1, 2, ..., 2p. The planar embedding of H*k* has the edges represented by column k of A drawn on the upper half of the sphere and the edges represented by column 2p+1-k drawn on the lower half of the sphere using the line of shortest distance between endpoints. The embedding of H2, for p=3, projected onto the plane, is shown in Figure 5(b).



(a) The Position of the Vertices on the Sphere for Hk



(b) The subgraph H2 of K12

Figure 5. Degree Constrained Partitioning of Kn

To prove that $t(K(m,p)) = [(m+5)/4]$, where $m = p$ (Theorem 2.1.14 part(a)), a planar partition of order $r+1$ is constructed for $K(4r+2,4r+2)$, for $r \geq 1$. Again, an array is used to construct the planar partition. Let the vertices in the first vertex set of $K(4r+2,4r+2)$ be numbered consecutively from 1 to $4r+2$, and the vertices in the second vertex set from $4r+3$ to $8r+4$. The subgraphs $H1$, $H2$, ..., $Hr+1$ in the planar partition are constructed using a $(2r+1)$ by $(r+1)$ array A where each element of A is a 1-tuple or 2-tuple of integers between 1 and $2r+1$ defined as follows:

a. $A(1,1) = (1)$,

b. $A(1,j) = (s, s+1)$, where $s = 3-2j \pmod{2r+1}$ and $2 \leq j \leq r+1$,

c. $A(i,j) = (i-2j+2) \pmod{2r+1}$, when $j = (i+1)/2$, or $j = (i+2)/2$, $2 \leq i,j \leq 2r+1$,

d. $A(i,j) = (s, s+1)$, where $s = i-2j+1 \pmod{2r+1}$, for $j \neq (i+1)/2$ or $j \neq (i+2)/2$, $2 \leq i,j \leq 2r+1$.

Example 2.2.4 For $r=1$, $4r+2 = 6$,

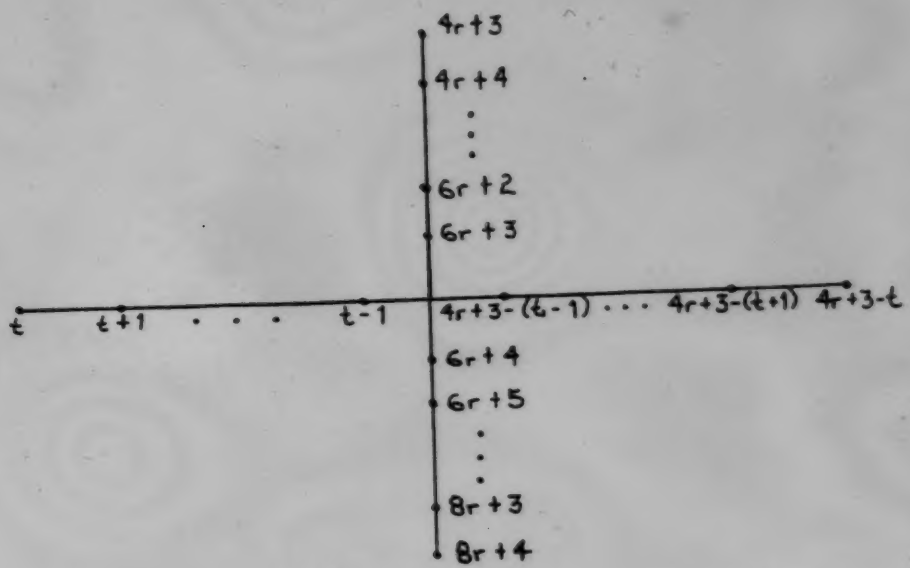
$$A = \begin{matrix} 1 & & 2\ 3 \\ 1\ 2 & & 3 \\ 2\ 3 & & 1 \end{matrix}$$

The planar embedding of each subgraph Hj, $j = 1, 2, \ldots, r+1$, is symmetric about both the vertical and horizontal axes of the plane. The position of each vertex of

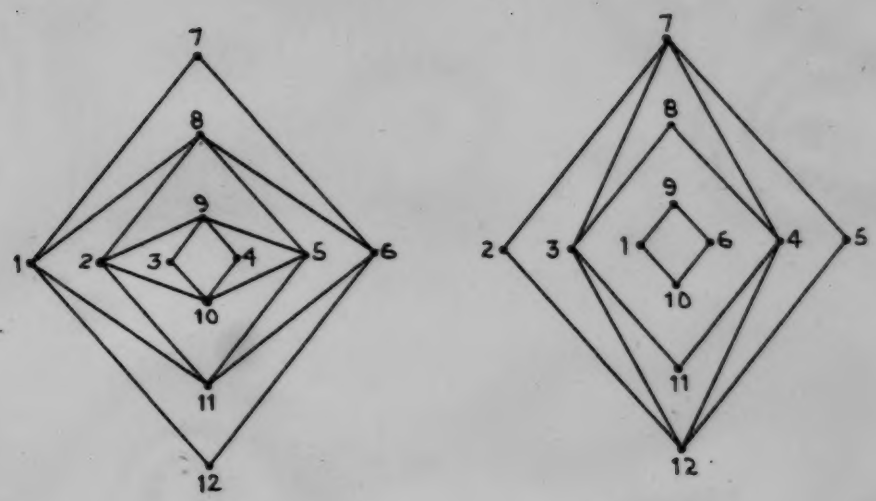
K(4r+2,4r+2) in the planar embedding of H*j* is shown in Figure 6(a), where t is the first entry of A(1,j). All the vertices to the left of the vertical axis are between 1 and 2r+1 and all the vertices to the right are between 2r+2 and 4r+2. Note that the vertices along the vertical axis are positioned independent of j while the position of each vertex along the horizontal axis depends on j. The edges of H*j* are represented by column j of A where the i'th row of A corresponds to the vertices 4r+2+i and 8r+5-i. If s is an entry of A(i,j), then the edges (4r+2+i, s), (4r+2+i, 4r+3-s), (8r+5-i, s), and (8r+5-i, 4r+3-s) are in H*j*. Figure 6(b) illustrates the embedding of the subgraphs H*1* and H*2* in the planar partition for K(6,6). The subgraphs are constructed using the array A from example 2.2.4.

The proof that $t(K(m,p)) = [(p+3)/4]$ for $m \leq p-2$ (Theorem 2.1.14 part (b)) involves constructing a planar partition of order $[(p+3)/4]$ for K(m,p), where $p = 4r+4$, $r \geq 1$, and $m \leq 4r+2$. Let the m vertices of the first vertex set of K(m,4r+4) be numbered from 4r+5 to 4r+4+m, and let the 4r+4 vertices of the second vertex set be numbered from 1 to 4r+4. Also, let $s = \{m/2\}$ (the smallest integer greater than or equal to m/2). An s by (r+1) array A is constructed and used to form the planar partition of K(m,4r+4). Each element A(i,j) is a pair of consecutive integers between 1 and 2r+2. For $1 \leq i \leq s$, and $1 \leq j \leq r+1$,

$$A(i,j) = (h, h+1) \pmod{2r+2}, \text{ where } h = i-2j+2.$$



(a) The Vertices of H*j* for K(4r+2,4r+2) in the Plane



(i) H1 (ii) H2

(b) A Planar Partition for K(6,6), with d(v) ≤ 4

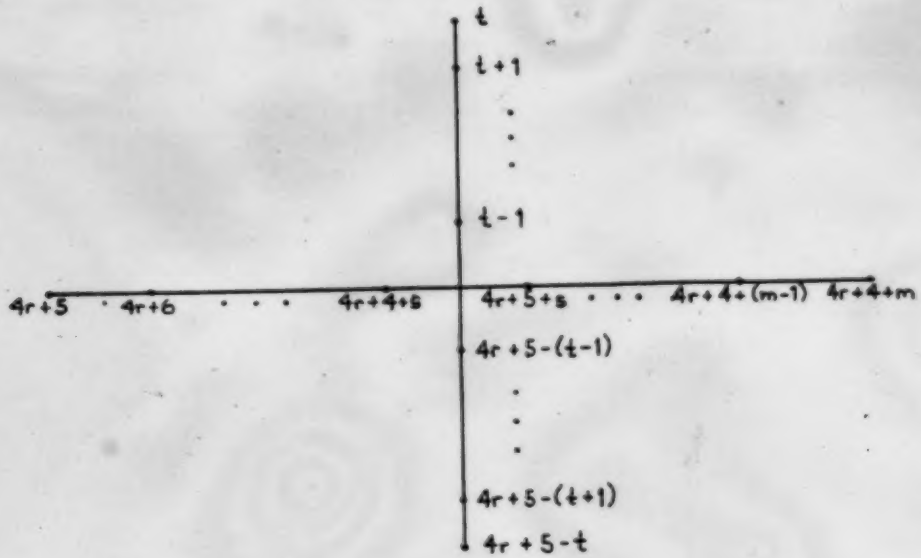
Figure 6. Degree Constrained Partitioning of K(4r+2,4r+2)

Example 2.2.5 For r = 2, 4r+4 = 12, and m = 10

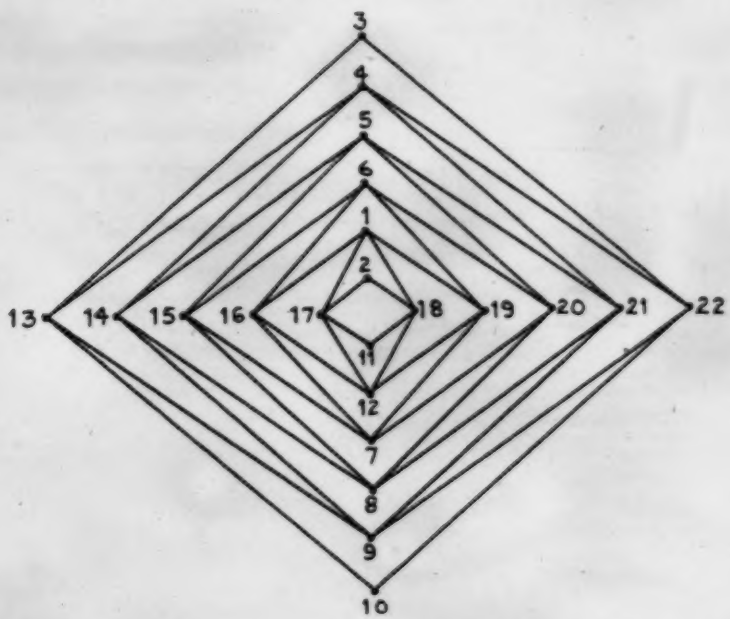
$$A = \begin{matrix} 1\ 2 & 5\ 6 & 3\ 4 \\ 2\ 3 & 6\ 1 & 4\ 5 \\ 3\ 4 & 1\ 2 & 5\ 6 \\ 4\ 5 & 2\ 3 & 6\ 1 \\ 5\ 6 & 3\ 4 & 1\ 2 \end{matrix}$$

Each subgraph H*j*, j = 1, 2, ..., r+1, has the same vertices as K(m,4r+4). In the planar embedding of H*j* its vertices are placed in the plane as depicted in Figure 7(a), where t is the first entry in A(1,j). The vertices numbered between 1 and 2r+2 are place above the horizontal axis and the vertices numbered between 2r+3 and 4r+4 are placed below the horizontal axis. Note that the position of the vertices on the horizontal axis is independant of j and the position of the vertices on the vertical axis varies with j.

To form H*j*, the j'th column of A, j = 1, 2, ..., r+1, is used. For i = 1, 2, ..., s, if A(i,j) = (h, h+1), then the edges (h, 4r+4+i), (h+1, 4r+4+i), (4r+5-h, 4r+4+i), and (4r+5-(h+1), 4r+4+i) are in H*j*, and if m is even or i ≥ 2, then the edges (h, 4r+5+m-i), (h+1, 4r+5+m-i), (4r+5-h, 4r+5+m-i), and (4r+5-(h+1), 4r+5+m-i) are also in H*j*. Figure 7(b) shows the planar embedding of the subgraph H*3* for K(10,12) constructed using the array A from example 2.2.5.



(a) The Vertices of Hj for K(m,4r+4) in the Plane



(b) The subgraph H3 for K(10,12), with d(v) ≤ 4

Figure 7. Degree Constrained Partitioning of K(m,4r+4)

Chapter 3

Planarity Algorithms

Planarity algorithms determine whether or not a graph is planar (of thickness one). Two such algorithms are discussed in this chapter. The algorithms form the basis for two of the edge partitioning algorithms that are proposed in Chapter 4. The first algorithm is a path finding algorithm from Hopcroft and Tarjan [Hop 74], the second a pq-tree algorithm from Booth and Lueker [Boo 76].

3.1 A Path Finding Planarity Algorithm

A forerunner of this planarity algorithm is described in Auslander and Parter [Aus 61]. Their algorithm is modified in Tarjan [Tar 71] to eliminate the possibility of an infinite loop and obtain a linear run time. Tarjan's version of the algorithm is put into a more concise form in Hopcroft and Tarjan [Hop 74]. It is this concise form that is described here.

Briefly, the algorithm determines whether a biconnected graph G is of thickness one by embedding a cycle C of G in a plane and, then attempting to embed the remaining portion of G around C such that the embedding is planar. If G is planar, then, as a result of the Jordan Curve Theorem [Tar 71], a planar embedding is possible starting with any cycle of G. Using depth first traversal and a special

ordering of the adjacency lists of G, the algorithm constructs a planar embedding of G (or decides that such an embedding does not exist) in linear time and space. The time complexity of the algorithm is O(e); however, only graphs with $e \leq 3n-6$ need to be tested for planarity, yielding an O(n) time complexity.

3.1.1 A Special Ordering of the Adjacency Lists of G

A depth first traversal, [Tar 71], of the vertices and edges of an undirected, simple graph G with n vertices enables the restructuring of G into a form more conducive to planarity testing.

First, a depth first traversal of G can be used to assign to each vertex v a preorder number g(v) (an integer value between 1 and n) which is the traversal number of v. Secondly, the edges of G can be classified (using the terminology of Reingold, Nievergelt, and Deo [Rei 77]) into two types: **tree** edges and **back** edges. An edge (v,w) of G is classified as a tree edge if w has not been visited prior to the traversal of the edge. Otherwise, (v,w) is classified as a back edge. A tree edge (v,w) has the property that $g(v) < g(w)$, and for a back edge (v,w), $g(v) > g(w)$. Finally, a depth first traversal of G can be used to direct its edges in the direction of traversal.

The planarity algorithm performs a depth first traversal of G to renumber each vertex v in V using g(v), to classify each edge of G as a tree edge or a back edge, and

to direct each edge. In addition, for each vertex v, two lowpoint values: LOWPT1(g(v)) and LOWPT2(g(v)) are determined. These lowpoint values are the smallest and second smallest values of g respectively that are assigned to vertices reachable from v via a path traversed during the depth first traversal. Figure 8 shows the LOWPT1 and LOWPT2 values for the vertices of a graph where each vertex v has already been relabelled as g(v). The edges of the traversed graph G are sorted into nondecreasing order by key, where the key for each edge is computed by the function Þ defined as:

$$
\begin{aligned}
Þ[(v, w)] &= 2*w && \text{if } (v, w) \text{ is a back edge} \\
&= 2*\text{LOWPT1}(w) && \text{if } (v, w) \text{ is a tree edge} \\
& && \text{and LOWPT2}(w) \geq v \\
&= 2*\text{LOWPT1}(w) + 1 && \text{if } (v, w) \text{ is a tree edge} \\
& && \text{and LOWPT2}(w) < v.
\end{aligned}
$$

Finally, ordered adjacency lists (with edges ordered in nondecreasing values of Þ), are constructed from the sorted edge list. Once the adjacency lists are constructed, an embedding procedure determines the planarity of G by attempting to systematically embed G in a plane. The systematic embedding is performed via a depth first traversal of the restructured graph G. The traversal commences at vertex 1 and uses the ordered adjacency lists of G.

1. LOWPT1(3) = 1, LOWPT2(3) = 3
2. LOWPT1(4) = 1, LOWPT2(4) = 3
3. LOWPT1(5) = 1, LOWPT2(5) = 3

Figure 8. Reachable Vertices

3.1.2 The Embedding Procedure

Assume that the vertices of G have been numbered via a depth first traversal of G and that the edges of G are directed and classified and the adjacency lists ordered as described in the previous section.

Definition 3.1.1

A cycle C of a graph G is a path of G such that the first vertex of the path is the same as the last vertex.

Paths and cycles in the embedding procedure are limited to a sequence of tree edges and one back edge. A path from v to w will be represented by p: v -->* w.

Definition 3.1.2

A **segment** S of a cycle C is a connected subgraph of G having no edges in common with C and consisting of either:

1. a single back edge with both vertices on C (a **trivial** segment), or
2. one tree edge (s, v), with s on C and v not on C, plus the directed tree rooted at v along with the set of back edges emanating from the tree (a **nontrivial** segment).

If a cycle C is removed from G, the subgraph G - C is the collection of segments induced by C. Every edge of G - C belongs to one and only one such segment.

Definition 3.1.3

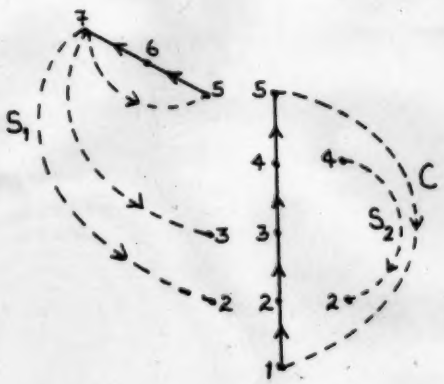
The vertices common to a cycle C and a segment S of C are called the **vertices of attachment** of S (see Figure 9).

It is advantageous to distinguish two vertices of attachment of a segment S from the others.

Definition 3.1.4

The highest numbered vertex of attachment of a segment S is called the **start** vertex of S and the lowest numbered vertex of attachment is called the **finish** vertex.

(a) The graph G.



(b) G decomposed into a cycle C: 1->2->3->4->5-->1 and the segments induced by C.
S_1 is a nontrivial segment with vertices of attachment {2,3,5} where 2 is the finish vertex, and 5 is the start vertex.
S_2 is a trivial segment and the section t_2 for S_2 is t_2: 2->3->4.

Figure 9. The Decomposition of a Graph

Definition 3.1.5 (Even [Eve 79])

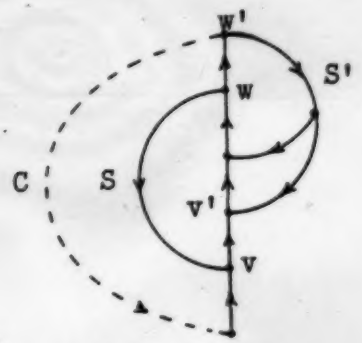
Two segments S and S' **interlace** if either

1. There are four distinct vertices v, w, v', w' on C such that v and w are vertices of attachment of S and v' and w' are vertices of attachment of S' and the four vertices appear in cyclic order v, v', w, w' (as in Figure 10(a)), or
2. S and S' have three vertices of attachment in common (as in Figure 10(b)).

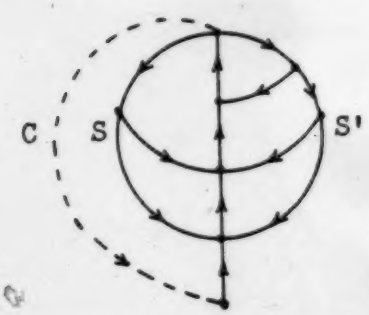
If two interlacing segments are embedded on the same side of C then they **overlap**. Two segments that do not interlace can be embedded on the same side of C with no loss of planarity.

There are some important conceptual ideas underlying Hopcroft and Tarjan's planarity algorithm. To construct a planar embedding of G, a cycle C of G is determined. The cycle C (containing vertex 1) is the first cycle traversed during the depth first traversal of G. The construction process consists of embedding C in a plane and then systematically attempting to embed each segment in the plane in such a way that no two segments overlap. If such an embedding is not possible, then G is nonplanar.

Given a segment S of C let t represent the section of tree edges of C between the finish vertex f and the start vertex s of S, t: f ->* s, (see Figure 9). The process of embedding each segment S with respect to C consists of performing two planarity tests.



(a) v, v', w, w'



(b) Three common vertices

Figure 10. S and S' Interlace

1. The first planarity test, under the assumption that S U C is planar, checks for interlace between S and the segments embedded before S. If two interlacing segments S' and S" are found that interlace with S, then G is nonplanar, otherwise, it is possible to embed S so that it does not overlap with the previously embedded segments. Thus, if S U C is planar, then S U C U {previously embedded segments} is planar.
2. The second planarity test determines whether S U C is planar. This test is broken down into two parts:

(a) Testing the planarity of S U t (ignoring C - t), and

(b) Testing the planarity of (S U t) U (C - t).

Algorithm 3.1.1 and Algorithm 3.1.2 illustrate the general flow of the planarity algorithm. The first algorithm assumes that G is a biconnected graph and returns the value PLANAR if G is planar and NONPLANAR otherwise. The second algorithm assumes that G has the structure obtained from a depth first traversal and has ordered adjacency lists, and that C is a cycle of G. The second algorithm returns the value TRUE if G can be embedded in the plane, FALSE otherwise.

Algorithm 3.1.1 (PLANARITY(G))

1. Perform a depth first traversal of G; directing and classifying the edges of G and renumbering its vertices.
2. Construct the specially ordered adjacency lists for G.
3. Determine the cycle C of G (by depth first traversal of G starting at vertex 1) and embed C.
4. If EMBED(G,C), then return PLANAR.
5. Else, return NONPLANAR.

Algorithm 3.1.2 (EMBED(G,C))

1. For each segment S of C do
   - 1.1 Generate a path p of S (by depth first traversal).
   - 1.2 If S passes the first planarity test, then
     - 1.2.1 Embed p in the plane.
     - 1.2.2 If S is a nontrivial segment, then
       - 1.2.2.1 Set C' = p U t.

Comment: Perform the second planarity test

1.2.2.2 If (¬EMBED(S U t, C')) or
((S U t) U (C - t) is nonplanar),
then return FALSE.

Comment: Otherwise S passes the second planarity test and, in the recursive call to EMBED, S has been completey embedded.

1.3 Else, return FALSE.

Comment: S fails the first planarity test.

2. Return TRUE.

Because the second planarity test is performed by a recursive call to the embedding procedure it is necessary to break the test into the two parts shown. If the test were not broken down, it would be possible, during the recursive call, to infinitely decompose the graph S U C into the cycle C and the segment S.

Note that the second test need not be performed for trivial segments because S U C is always planar. For nontrivial segments, on the other hand, both tests must be performed. Also note that the use of depth first traversal to systematically embed the segments of G, means that each segment is completely embedded before the embedding of a new segment starts.

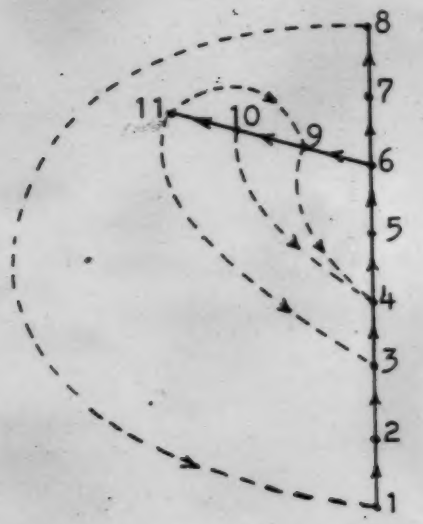
The embedding of a segment S of C commences with the traversal of a path p of S. This first path p: s -->* f traversed in S always has the property that the vertices s

and f are the start and finish vertices respectively of S. Henceforth, the first path p of a segment S is referred to as the **leading** path of S. Once p has been identified, the first planarity test can be performed on S because, as is shown later, only the leading path of S is required to perform this test. If S passes the test, then p is embedded in the plane and the procedure performs the second planarity test on S. For convenience p is embedded on the inside of C and all previously embedded segments are shifted about C as required to maintain a planar embedding.

Part (a), of the second planarity test is performed via a recursive call to the embedding procedure with S U t as input. Note that the subgraph S U t is already numbered, directed, and has the specially ordered adjacency lists described in section 3.1.1. Therefore, the embedding procedure can commence with the embedding of S U t without the preprocessing which was required for G. In addition, since a cycle C' comprised of t: f ->* s and the leading path p: s -->* f of S is already embedded in the plane, the embedding procedure only has to embed the segments induced by C' to determine whether S U t is planar. While the segments of C' are being embedded the rest of G and, in particular, the section C - t of the cycle C is ignored. If each segment can be embedded in the plane, then S U t is planar, and part (b) of the second planarity test (testing the planarity of (S U t) U (C - t)) is performed. In part (b), the section of edges C - t is considered. Because of

the nature of the cycle C', the segments induced by C' in the graph (S U t) U (C - t) are the segments induced by C' in S U t plus the segment C - t (see Figure 11). Thus, if C - t passes the two planarity tests, then (S U t) U (C - t) is planar. Since ((C - t) U C' = C U p is planar), the second test is trivially true. Thus the procedure only has to check for the existence of two segments of C' that interlace with C - t and with each other (the first planarity test). Since C - t interlaces with a segment if and only if that segment has at least one vertex of attachment v on t such that v ≠ s or f, the segment C - t is embeddable (and S U C is planar) if and only if all segments of C' with vertices of attachment on t are embedded on the inside of C' or can be shifted (without overlap) to the inside of C'.

The first planarity test on a segment S is by far the most interesting of the two tests and potentially the most difficult to perform. It is possible to efficiently determine which embedded segments interlace with S partly because of the order in which the segments are embedded (the order imposed by depth first traversal and the ordered adjacency lists of G). The order in which segments are embedded and the importance of that order are indicated in the following lemmas and theorem.



C: 1 ->* 8 --> 1
p: 6 -> 9 -> 10 -> 11 -> 3
t: 3 -> 4 -> 5 -> 6
C-t: 6 -> 7 -> 8 --> 1 -> 2 -> 3
C': t U p

Figure 11. S U C

Lemma 3.1.1 Let S and S' be segments of the same cycle C, let s' and f' be the start and finish vertices respectively of S', and let s and f be the start and finish vertices respectively of S.

(1) If S' is embedded before S, then s' ≥ s. That is, segments of a cycle are embedded in nonincreasing order of their start vertices.

(2) If s' = s, and f' < f, then S' is embedded before S.

Part (1) of Lemma 3.1.1 holds because G is traversed depth

first during the embedding process. Part (2) of the lemma holds because of the ordering of the adjacency lists of G and the depth first traversal of G.

Lemma 3.1.2 For all segments of a cycle C with the same start vertex and the same finish vertex, segments with only two vertices of attachment are embedded before segments with more than two vertices of attachment (see Figure 12).

Proof: Let S and S' be two segments of a cycle C with the same start vertex s and the same finish vertex f. Suppose that s and f are the only vertices of attachment of S and suppose that S' has at least one other vertex of attachment w. Let p be the leading path of S with (s, v) the first edge of p, and let p' be the leading path of S' with first edge (s, v'). The edges of G are sorted on their þ values, and according to the definition of þ:

$$þ[(s, v)] = 2*LOWPT1(v) = 2*f$$

and, since w is between s and f,

$$þ[(s, v')] = 2*LOWPT1(v') + 1 = 2*f + 1.$$

Therefore, (s, v) precedes (s, v') in G's ordered adjacency lists. Hence (s, v) is traversed before (s, v') and S is embedded before S'.

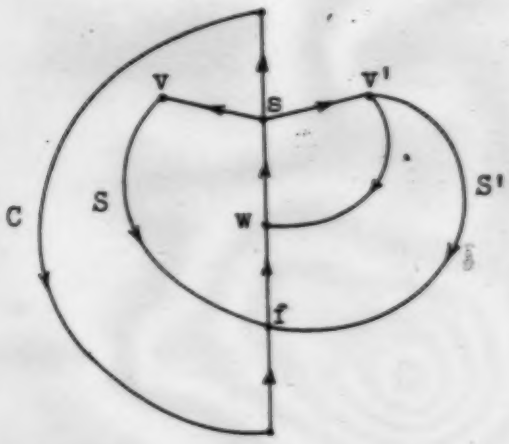


Figure 12. S is embedded before S'

Theorem 3.1.3 Let S and S' be two segments of a cycle C with S' embedded before S. Let s be the start vertex of S and f be its finish vertex. Let w' be the highest vertex of attachment of S' for which w' < s. Then, S and S' interlace if and only if w' exists and f < w'.

Proof:

A. To prove the if statement, there are two cases to consider.

Case 1. s' = s. (See Figure 13(a) and (b))

Since s' = s and S' is embedded before S, by Lemma 3.1.1, f' ≤ f. Also, since f < w', it follows that S' must have at least three vertices of attachment.

a. If f' < f, then f' < f < w' < s and therefore, S and S' interlace.

b. If f' = f, then, by Lemma 3.1.2, S must have at least three vertices of attachment as well.

1) If S and S' have three vertices of attachment in common they interlace by Definition 3.1.6.

2) If S and S' do not have three common vertices of attachment there exists a vertex of attachment w of S with $w < s$ and $w \neq w'$. Since $w \neq w'$ either the four vertices with cyclic order f, w', w, s' or the four vertices with cyclic order f', w, w', s may be used depending on whether $w > w'$ or $w < w'$ respectively, demonstrating that the segments interlace.

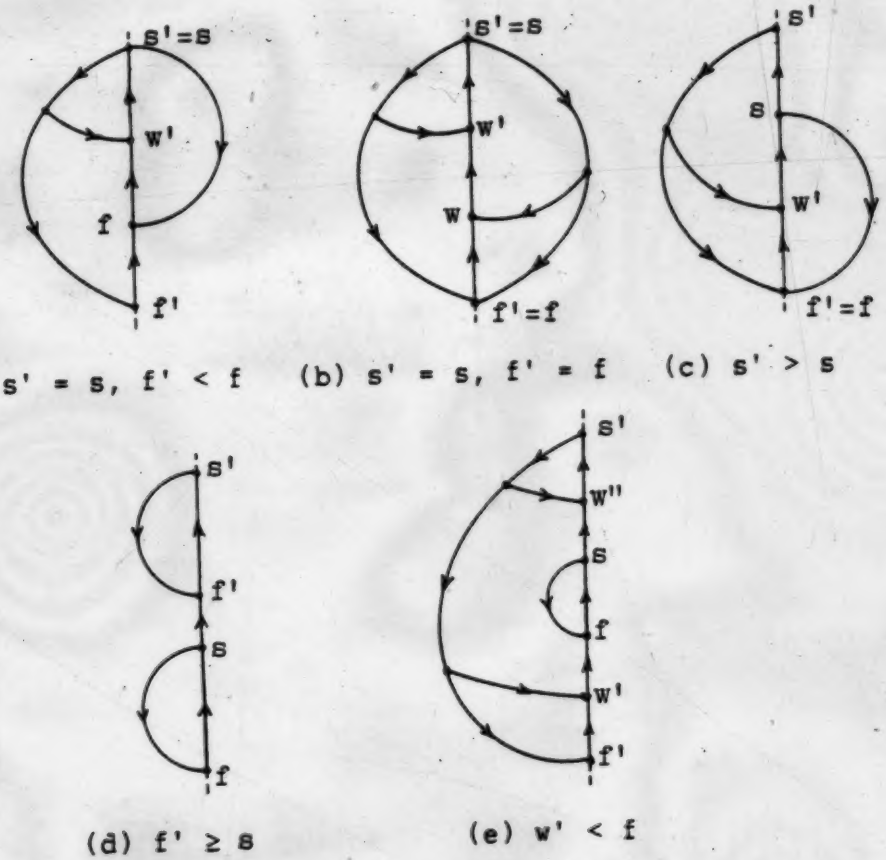
Case 2. $s' > s$.(See Figure 13(c))

From the hypothesis and this condition it follows that $f < w' < s < s'$ and therefore S and S' interlace.

B. The proof that S and S' interlace only if w' exists and $f < w'$ is accomplished through the proof of the contrapositive statement, that S and S' do not interlace if w' does not exist or $f \geq w'$.

Case 1. If w' does not exist, then $f' > s$ and the vertices of attachment of S occur on the cycle between the two consecutive vertices s' and f' of S'. Thus S and S' do not interlace (see Figure 13(d)).

Case 2. If $f \geq w'$, let w" be a vertex of attachment of S' such that $w" > w'$ and there is no vertex of attachment of S' between w' and w". By the definition of w' it follows that $w' \leq s \leq w"$ (see Figure 13(e)). If $f \geq w'$, then $w' \leq f < s \leq w"$.



(a) s' = s, f' < f (b) s' = s, f' = f (c) s' > s

(d) f' ≥ s (e) w' < f

Figure 13. S and S' interlace if and only if f < w'.

Since all vertices of attachment of S are between f and s it follows that S and S' do not interlace.

Thus, to determine if the segment S, for which the first planarity test is being performed, interlaces with an embedded segment S', only the start and finish vertices of S and the highest vertex of attachment w' of S' for which w' < s is required. Therefore, only the leading path of S has to be traversed before the first planarity test can be performed on S.

The embedding procedure must perform two other operations during the first planarity test in addition to checking if S and any previously embedded segments interlace:

(1) It must ensure that S and all segments interlacing with S do not overlap. This operation may require the shifting of segments from one side of C to the other.

(2) It must determine all segments affected by a shift in (1) and must ensure that they lie on the correct side of C to maintain a planar embedding.

To perform these two operations efficiently, the embedding procedure uses a data structure called a **block**.

Definition 3.1.6

Two segments of a cycle are said to **interact** if the embedding of one determines the embedding of the other.

If the segments S and S' interlace, then they interact by the above definition. Also, if S interlaces with S', and S' interlaces with S", then S and S" interact, even though S does not interlace with S".

Theorem 3.1.4 The relation **interact** is an equivalence relation over the set of embedded segments of a cycle C.

Definition 3.1.7

A **block** is an equivalence class of the relation **interact** over the set of embedded segments of a cycle. Each element of a block is a segment. The segment is represented

by those vertices of attachment that are end vertices of back edges (all vertices of attachment except the start vertex) of the segment.

A block has two important properties.

1. Repositioning any segment represented in a block requires the repositioning of all segments represented in that block but does not affect any segment not in that block.
2. After the planarity tests are performed on a segment, and it is embedded, the segment becomes an element of a unique block.

3.1.3 The Formation and Use of Blocks

Blocks enable the embedding procedure to efficiently perform both planarity tests on a segment S. Initially, the leading path p: s -->* f of S is traversed and the first planarity test is performed on S. If S interlaces with a segment S' represented in a block, then S interacts with all segments in that block. If S' must be shifted to embed S, then all other segments represented in that block must also be shifted. Most importantly though, shifting S' does not affect the position of any segment represented in a different block. Moreover, S fails the first planarity test if and only if S interlaces with two segments in the same block that interlace with each other. If S passes the first planarity test, then p is embedded in the plane on the

inside of C and a new block is formed which represents the union of the segment S and the blocks of segments that interact with S.

If S is a nontrivial segment, then the second planarity test is performed. First, the planarity of S U t is checked. The leading path p and the set of edges t form a cycle C', with the edges of S U t - C forming the segments of C' which are embedded. A set of blocks is formed during the embedding of these segments. The blocks in this set are the equivalence classes of the relation interact for the embedded segments of C'.

If S U t is planar, then the embedding procedure tests whether (S U t) U (C - t) is planar. The procedure must determine whether all the segments of C' in S U t with vertices of attachment on t, other than s and f, can be placed on the inside of C'. To accomplish this, each block of C' is checked for two interlacing segments with vertices of attachment v such that s > v > f. If two such segments are found, then S U C is nonplanar.

Let B represent the stack of blocks formed during the embedding process and let b = [bx, bz] represent a block on B, where bx and bz are sets of vertices. Each element of bx is a vertex of attachment representing a segment embedded on the inside of a cycle. Similarly, each element of bz is a vertex of attachment representing a segment embedded on the outside of the same cycle.

If a segment represented in bx is shifted to the outside of the cycle, then all segments represented in bx must be shifted to the outside and all segments represented in bz must be shifted to the inside of the cycle. When a recursive call is made to the embedding procedure to complete the embedding of a segment S, an end-of-stack marker is placed on top of the block stack to separate the blocks created for the segments of C' from the blocks created for the segments of C. The block just under the end-of-stack marker is the equivalence class for S on C.

Definition 3.1.8

Let p: s -->* f be a path and let p': s' -->* f' be the first path (traversed during the embedding process) having s as one of its vertices. The path p is called a **normal** path if f' < f; and p is called a **special** path if f' = f. If p is a special path, then p is said to be **attached** to p'.

Note that if p is a special path attached to p', then p is a path in the segment S' with leading path p'. The path p is traversed during the testing of the planarity of S' U t' and f is the finish vertex of some segment S" in S' U t'. Normally f would be placed in the equivalence class for S", however, since p is a special path, f is not placed in that block [Hop 74].

It is important to note that once the second planarity test is completed for a segment S all the vertices of attachment of S except (1) its finish vertex if its leading

path is special, and (2) its start vertex are placed in the equivalence class for S.

Algorithm 3.1.3 outlines the steps involved in the first planarity test. The algorithm uses Theorem 3.1.3 to determine if the segment S being tested interlaces with any previously embedded segment. For the algorithm to use the theorem, vertices are deleted from the blocks so that the largest vertex w' in each block is less than the start vertex s of S (blocks containing only vertices w' ≥ s are deleted from B). The deletions occur when the vertex s is visited prior to the traversal of the leading path p: s -->* f of S.

Algorithm 3.1.3 (The First Planarity Test)

1. Set b = [bx, bz] = [Ø, Ø], INTERLACE = true, and let p: s -->* f be the leading path of S.
2. While (the top block b' = [bx', bz'] on B is not an end-of-stack marker) and INTERLACE do
   2.1 Let x be the largest vertex in bx' and let z be the largest vertex in bz'. Let S' be the segment represented by x and let S" be the segment represented by z, then S' and S" interlace.
   Comment: Test whether S is embeddable with respect to the segments in the block b'.
   2.2 If x > f and z > f, then return FALSE.
   Comment: S' and S" interlace with S by Theorem 3.1.3 so S fails the first planarity test

and G is nonplanar.

2.3 Else, if x > f and z ≤ f, then

Comment: S and S' interlace but S and S" do not. Since S is to be embedded on the inside of C the segments represented in bx' are shifted to the outside of C and the segments represented in bz' are shifted to the inside.

2.3.1 bx = bz' U bx,

2.3.2 bz = bx' U bz,

2.3.3 delete b' from B.

2.4 Else, if x ≤ f and z > f, then

Comment: S and S' do not interlace but S and S" do. No shifting is required because with S embedded on the inside of C, S and S" will lie on opposite sides of C.

2.4.1 bx = bx' U bx,

2.4.2 bz = bz' U bz,

2.4.3 delete b' from B.

2.5 Else, INTERLACE = false.

Comment: x ≤ f and z ≤ f, so S does not interlace with either S' or S". No shifting is required and no more segments interlace with S.

Comment: S can be embedded on the inside of C and the block b contains all available vertices of attachment for all segments interacting with S.

3. If p is a normal path (not special), then set
   bx = bx U {f}.
4. If S is a nontrivial segment, then
   4.1 Place b on the block stack.
   4.2 Place an end-of-stack marker on top of b on B.
5. Else, **Comment:** S is a trivial segment.
   If b ≠ [Ø, Ø], then make b the top block on B.
6. Return TRUE.

If S is a nontrivial segment, then, after Algorithm 3.1.3 is performed, the planarity of S U C must be verified. First the planarity of S U t is checked, requiring a recursive call to the embedding procedure. An end-of-stack marker (step 4.2 of Algorithm 3.1.3) is already on the block stack for the call.

Once the planarity of S U t is ascertained the planarity of (S U t) U (C - t) = S U C is checked. If (S U t) U (C - t) is planar, then the topmost end-of-stack marker is removed from the block stack and all the vertices of attachment of S are placed in the block b' just under the marker (b' is the equivalence class for S). The verification of the planarity of (S U t) U (C - t) and the collection of the vertices of attachment of S into one block can be accomplished simultaneously because only vertices of S on t are required in both cases. Algorithm 3.1.4 performs the two operations. Before the algorithm is performed vertices v ≥ s are removed from the blocks above the topmost end-of-stack

marker. Blocks containing only vertices v ≥ s are removed from B. Thus the blocks remaining on B above the topmost end-of-stack marker contain only vertices v on t such that s > v > f.

Algorithm 3.1.4 (Part (b) of the Second Planarity Test)

1. Set b = [bx, bz] = [Ø, Ø].
2. While the top block b' = [bx', bz'] on B is not an end-of-stack marker do
   - 2.1 Let x' be the smallest vertex of bx' and let z' be the smallest vertex of bz'.
   - 2.2 If x' > f and z' > f, then return FALSE.
     Comment: S U C is nonplanar. That is, the two segments represented by x' and z' interlace with C - t and with each other.
   - 2.3 Else, if x' > f and z' ≤ f, then
     - 2.3.1 bx = bx' U bx (note that bz' = Ø in this case),
     - 2.3.2 delete b' from B.
   - 2.4 Else, if x' ≤ f and z' > f, then
     - 2.4.1 bx = bz' U bx (note that bx' = Ø in this case),
     - 2.4.2 delete b' from B.

   Comment: S U C is planar.
3. Delete the end-of-stack marker from B.

Comment: Combine b with the top block b' of B (the block formed when the leading path p of S was embedded).

4. Set bx = bx' U bx.
5. Set bz = bz'.
6. Delete b' from B.
7. If b ≠ [Ø, Ø], then make b the top block on B.
8. Return TRUE.

If the block formed by Algorithm 3.1.3 for S remains on B, then Algorithm 3.1.4 will successfully run to completion provided S U C is planar.

3.1.4 A Correction to the Implementation of the Embedding Procedure

In the PATHFINDER implementation of the embedding procedure [Hop 74], each block b is an ordered pair (x, y), where x and y are pointers to a vertex stack. The entries x and y point to the smallest vertex of attachment of the block representing a segment or segments embedded on the inside and outside of a cycle respectively. A zero placed on the vertex stack serves as the end-of-stack marker for the block stack. If there are no edges in the block on the inside (outside) of the cycle, then x (y) is assigned the value 0. However, a block (0, 0) is created if a segment is not constrained by any segments previously embedded and the leading path of the segment is a special path. Thus, the pair (0, 0) serves as the block for a segment. The erroneous code in Hopcroft and Tarjan's algorithm is

```
While (x, y) on B has
     ((STACK(x) ≥ v) or (x = 0)) and
     ((STACK(y) ≥ v) or (y = 0))
   do delete (x, y) from B;
```

This code allows the block (0, 0) for a segment S to be deleted from the block stack prior to the end of the recursive call made to embed S, thus steps 4, 5, and 6 of Algorithm 3.1.4 will not execute correctly once S has been embedded. Since (0, 0) should not be deleted before the end of the recursive call correct code is:

```
While (x, y) on B has
     (((STACK(x) ≥ v) and
       (STACK(y) ≥ v)      ) or
      ((STACK(x) ≥ v) and
       (y = 0)             ) or
      ((x = 0)          and
       (STACK(y) ≥ v)         ) )
   do delete (x, y) from B;
```

3.2 A pq-tree Planarity Algorithm

The pq-tree planarity algorithm from Booth and Lueker [Boo 76] is an O(n) version (in time and space, where $e \leq 3n-6$) of a vertex embedding algorithm from Lempel, Even, and Cederbaum [Lem 67]. It accepts as input a biconnected graph G. First, the vertices of G are given a special

numbering, then the numbered vertices are visited in ascending order by an embedding procedure.

3.2.1 An st-Numbering of G

The first step of the planarity algorithm renumbers the vertices of the biconnected graph G using a special numbering, called an **st-numbering.**

Definition 3.2.1

An **st-numbering** is an integer valued function f defined on the vertices of G = (V, E), where |V| = n, as follows:

(1) Take any edge (s, t) in G and let

f(s) = 1, and

f(t) = n.

(2) For any vertex v in V, $1 \leq f(v) \leq n$.

(3) For any vertices v and w in V, if $v \neq w$, then $f(v) \neq f(w)$.

(4) For any vertex v in V, if $v \neq s$ or t, there exist vertices u and w adjacent to v such that $f(u) < f(v) < f(w)$.

Lempel, Even, and Cederbaum [Lem 67] prove that an st-numbering can be obtained for any biconnected graph G.

Even and Tarjan [Eve 76] developed a linear time (O(e)) algorithm for computing an st-numbering using depth first traversal. Given an edge (s,t) in G , a depth first spanning tree of G is found with (t,s) the first edge of the tree. The vertices of G are numbered in preorder from 1 to n, with

t receiving the number 1 and s receiving the number 2. In addition, for each vertex with preorder number v, the smallest preorder numbered vertex, LOW(v), reachable from v, is computed and all edges not in the depth first spanning tree are labelled as back edges. In the remainder of this section, the label v refers to the preorder number of a vertex and f(v) refers to the st-number of that vertex.

A second depth first traversal of G is performed, to assign each vertex an st-number. First the vertices 1 and 2 are marked "old" and placed on a vertex stack with vertex 2 placed on top of vertex 1. Old vertices are vertices which are or were at one time on the vertex stack. Vertices which have never been on the stack are "new" vertices. The vertex stack, during the course of the st-numbering procedure, contains vertices which have not been assigned st-numbers. The vertex 2 is the first vertex to receive an st-number and so receives the st-number 1. Every other vertex v of G is assigned an st-number in the following way. When v reaches the top of the vertex stack, it is removed from the stack, and all unexplored paths from v to an old vertex u are traversed in an effort to obtain new vertices to add to the stack. The paths traversed have the form p = (v,w1,w2,...,wk,u), where k > 0, u is an old vertex, and w1, w2, ..., wk are new vertices. First, paths p = (v,w1,w2,...,wk,u) are explored where (v,w1) is a tree edge and u = LOW(w1) = LOW(w2) = ... = LOW(wk). The new vertices of p are marked old and added to the vertex stack in order

wk, wk-1, ..., w1. Next, paths p = (v,w1,w2,...,wk,u) are explored where (v,w1) is a back edge and the remaining edges of p are tree edges on the tree path from v to w1. Again, the new vertices of p are marked old and added to the vertex stack in order wk, wk-1, ..., w1. After all paths from v are explored, v is assigned the next consecutive st-number. The process of path exploration is repeated for each top vertex on the vertex stack until the stack is empty.

The paths explored by the st-numbering procedure have certain properties that are essential to the correct assignment of an st-numbering f. Let p = (v,w1,...,wk,u) be any path explored by the st-numbering procedure, then

(1) p is a simple path, that is, $v \neq u$,

Proof:

(i) If (v,w1) is a tree edge, then u = LOW(w1) and $u < v$ because G is biconnected and v is never vertex 1.

(ii) If (v,w1) is a back edge, then (wk,u) is a tree edge and $v \neq u$ because all tree edges from v are explored before any back edges are.

(2) v and u of the path p are old vertices and all the other vertices of p are new.

(3) The last vertex u of p has not been assigned an st-number at the time p is explored.

Proof: Assume that the vertex u has already been assigned an st-number f(u). Then, the edge (u,wk) would already have been traversed and wk would not be a new

vertex, a contradiction.

Theorem 3.2.1 The st-numbering procedure assigns an st-numbering f to the vertices of G.

Proof:

1. All the vertices v of G receive an st-number f(v) because G is connected.
2. Each vertex is assigned a unique number because a vertex is placed on the vertex stack only once (when its new).
3. f(2) = 1 because 2 is the first vertex to receive an st-number. The preorder number of the vertex s is 2, thus vertex s receives st-number 1.
4. f(1) = n

   Proof: If f(1) < n, then some vertices of G have not been on any paths explored before 1 reaches the top of the vertex stack. But, that would mean that 1 is an articulation point in G, a contradiction. The preorder number of the vertex t is 1, thus vertex t receives st-number n.
5. For any vertex v ≠ 2 or 1, there exist vertices u and w adjacent to v such that f(u) < f(v) < f(w).

   Proof: When v receives an st-number f(v), it is an old vertex. The vertex v becomes old when a path p is explored with v as an interior vertex. Thus v has vertices u and w on either side of it in the path such that w is placed on the vertex stack (or was on the vertex stack) before v, and u is placed on the stack

after v, or u is the vertex being assigned an st-number when p is explored. As a result, u is assigned an st-number f(u) before v so f(u) < f(v), and w is assigned an st-number f(w) after v so f(v) < f(w).

Once the vertices of G have been assigned st-numbers (see Figure 14), each vertex is relabelled using its st-number, and each edge is directed from its lower st-numbered vertex to its higher st-numbered vertex. If G is an st-numbered graph, let V_k be a subset of V with $V_k = \{1, 2, \ldots, k\}$ and $G_k = (V_k, E_k)$, where E_k is the set of all edges of G with both vertices in V_k [Eve 79]. If G is planar, then so is G_k. Let $\bar{G}$ be a planar embedding of G and let $\bar{G}_k$ be the induced planar embedding of G_k. The following are true:

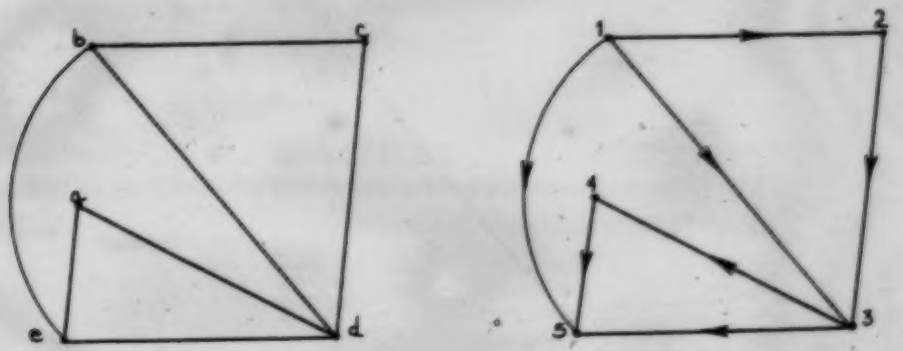
(1) The vertex with st-number 1 is the only source in the graph, that is, it is the only vertex having no directed edges entering it.

(2) The vertex with st-number n is the only sink in the graph, that is, it is the only vertex having no directed edges leaving it.

(3) G_k is connected.

(4) all the vertices and edges of $G - G_k$ lie in one face of $\bar{G}_k$.

Proof: All the vertices of $G - G_k$ have st-numbers greater than k and there is only one sink vertex n in $\bar{G}$. Since G is connected and n is the only sink of G,



G

An st-Numbered version of G

Figure 14. An st-numbering of a Graph

there is a directed path between any vertex v in G - G*k* and the vertex *n*. If the vertices of G - G*k* lay in two or more faces of ~G*k* there would be at least one edge crossing from one face of ~G*k* to another face contradicting the fact that ~G is a planar embedding of G.

3.2.2 The pq-tree Embedding Procedure

Henceforth, assume that G is an st-numbered, directed graph and that the vertices of G are labelled with their st-numbers. The embedding procedure attempts to construct a planar embedding of G. First, vertex 1 is visited and all its incident edges (edges of the form (1,j) with j > 1) are placed in the plane. The edges, called **virtual edges**, are directed downwards (for expediency) from vertex 1. Vertex 1 is a **real vertex** and all the vertices j > 1 in the plane are **virtual vertices** [Eve 79].

Definition 3.2.2

A **real vertex** is a uniquely numbered vertex in the plane which has been visited by the embedding procedure.

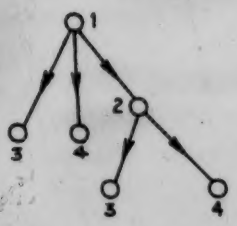
Definition 3.2.3

A **virtual vertex** is a vertex in the plane which has not been visited by the embedding procedure. There may be many virtual copies of the same vertex in the plane.

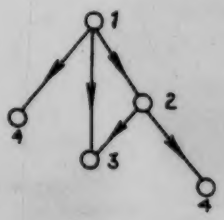
Definition 3.2.4

A **virtual edge** is an edge in the plane that has one real vertex and one virtual vertex as endpoints. A virtual edge becomes a **real edge** when the virtual vertex becomes real.

After vertex 1 and all the edges emanating from it are placed in the plane, the procedure attempts to merge all virtual vertices numbered 2. The virtual copies of any vertex can be merged if they are adjacent or can be made adjacent on the periphery of the embedding. If the virtual copies cannot be made adjacent, then G is nonplanar and the procedure halts. If the virtual copies of vertex 2 are merged, then a single vertex 2 is created and made real, the edges of the form (j,2) with $j < 2$, become real edges, and edges of the form (2,j) with $j > 2$, are added to the plane as virtual edges and directed upwards from 2 to j. The merging process is repeated for vertex numbers 3, 4, ..., n-1 (see Figure 15). If all the merging attempts are



(a) Vertices 1 and 2 have been visited



(b) Vertex 3 is visited and made real

Figure 15. Real and Virtual Vertices

successful then G is planar.

Let W_k be the set of all virtual vertices and let F_k be the set of all virtual edges lying in the plane after the creation of the real vertex numbered k, $1 \leq k \leq n-1$. V_k and G_k as defined before correspond to the real portions of the embedding. Then

$W_k = \{j: j$ is a virtual vertex in the embedding, $j > k\}$, and

$F_k = \{(i,j): i \leq k$ (i is in V_k), and $j > k$ (j is in W_k)$\}$.

Let $\bar{B}_k$ be the embedding in the plane after the creation of the real vertex numbered k. Then $B_k = G_k \cup (W_k, F_k)$ and $\bar{B}_k$ is the planar embedding of B_k. If G is planar, then, by property 4 of the st-numbering, all vertices and edges of $G - G_k$ are located in one face of any planar embedding of G_k. In particular, there is a planar embedding of G_k where the vertices and edges of $G - G_k$ are in its outer face. If G is nonplanar, then for some k this will not be true. Now,

B_k contains a planar embedding of G_k where all the vertices and edges of $G - G_k$ (represented by (W_k, F_k)) are in its outer face. If the virtual copies of a vertex k could not be made adjacent on the periphery of the embedding but were merged nevertheless, then all the vertices and edges of $G - G_k$ would not lie in one face of the embedding of G_k implying that G is nonplanar.

The embedding procedure uses a **pq-tree** data structure, developed by Booth [Boo 76], to achieve a linear run time for the embedding procedure. A pq-tree is comprised of three types of nodes: leaf-nodes, p-nodes, and q-nodes. The leaf-nodes represent the virtual edges and the p-nodes and q-nodes represent the real sections of the embedding.

Definition 3.2.5

1. A **leaf-node** is a terminal node in the tree.
2. A **p-node** is an interior node that has at least two children.
3. A **q-node** is an interior node that has at least three children.

The p-nodes and q-nodes are distinguished by ordering restriction imposed on their children and by the transformations allowed on their children. The children of a p-node are not ordered and any permutation of the children is permissible. The children of a q-node have a strict left-to-right order and only a reflection of that order is allowed. The children of a q-node are chained in a sibling

chain because of their ordering. The two children at either end of the chain are called endmost children, while the other children are referred to as interior children.

The embedding $\tilde{B}_k$ is represented by a pq-tree T_k as follows:

1. The leaf-nodes of T_k are in one-to-one correspondence with the edges in F_k.
2. A p-node in T_k represents either a real vertex in G_k or a connected component of G_k which has virtual edges emanating from at most two vertices.
3. A q-node in T_k represents a connected component of G_k which has virtual edges emanating from three or more vertices.

In the figures depicting pq-trees, a circle represents a p-node, a rectangle represents a q-node, a square represents a leaf-node, and a triangle represents any type of node and the subtree below that node. The relationship between $\tilde{B}_k$ and T_k is illustrated in Figure 16. Henceforth, the embedding procedure is described in terms of the pq-trees T_k. The corresponding embeddings $\tilde{B}_k$ are only mentioned when necessary.

Let S_{k+1} be the set of leaf-nodes in T_k corresponding to the virtual edges in F_k with virtual vertex k+1.

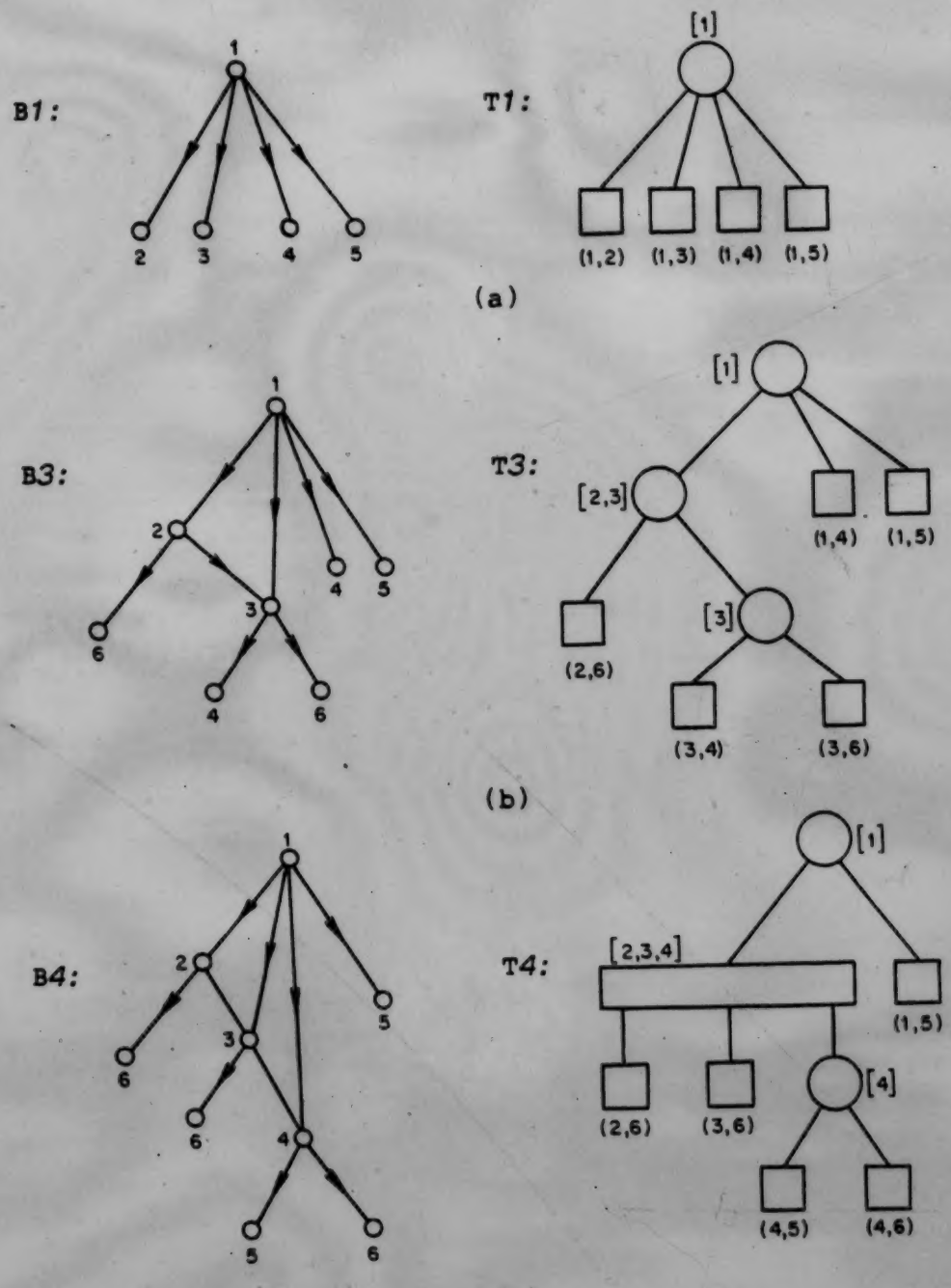


Figure 16. Embeddings and the Corresponding pq-trees

Definition 3.2.6

An element of S_{k+1} is called a **pertinent leaf-node** of T_k.

Definition 3.2.7

A **pertinent node** of T_k is a node which has at least one pertinent leaf-node as a descendant.

For example, in Figure 16(b) $S_4 = \{(3,4), (1,4)\}$, the leaf-nodes (3,4), and (1,4) are pertinent leaf-nodes in T_3, and the nodes (3,4), (1,4), [3], [2,3], and [1] are the pertinent nodes of T_3.

There is one modification operation M(T,x) that is performed on a node x and its children in the pq-tree T. M(T,x) is performed in two parts.

(1) An attempt is made to match x and its children to a template pattern.

(2) If a match is found, then x and its children are modified using a corresponding template replacement.

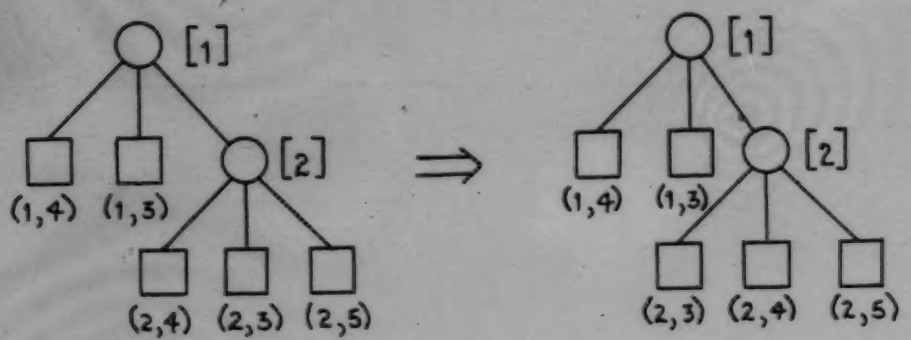
The possible template patterns and their corresponding template replacements are described in Booth and Lueker [Boo 76].

If G is planar, then for each iteration k+1 of the embedding procedure, k = 1, 2, ..., n-2, (vertex k+1 of G is visited) either all the pertinent leaf-nodes of T_k are next to each other on its periphery, or T_k can be transformed by modification operations, so that all the pertinent

leaf-nodes are adjacent (see Figure 17(a)). If G is nonplanar, then during some iteration of the procedure there is always at least one nonpertinent leaf-node between two pertinent leaf-nodes (see Figure 17(b)) regardless of the transformations applied to T_k.

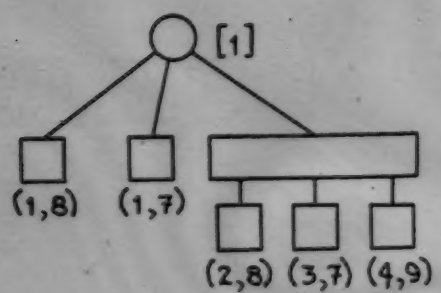
Thus merging virtual vertices numbered k+1 in B_k is possible only if the nodes of S_{k+1} can be next to each other on the periphery of T_k. Note that a nonpertinent leaf-node lying between two pertinent leaf-nodes represents a *possible* cross-over of edges in an embedding of G. If the pq-tree can be transformed to eliminate these possible cross-overs and no new cross-overs are formed, then G may be planar. If it is not possible to eliminate all the cross-overs, then, by property 4 of the st-numbering, a planar embedding of G cannot be constructed.

Note that if the vertices of G do not have an st-numbering, then the inability to eliminate all possible cross-overs would not necessarily mean that G is nonplanar as will be shown in section 4.4.2. An st-numbering of the vertices of G also means that as each vertex k+1 is visited, for consecutive values of k, $S_{k+1} \neq \emptyset$; that is, there is at least one virtual vertex numbered k+1 in B_k and B_{k+1} is connected. Therefore, for all k, $1 \leq k \leq n-2$, T_{k+1} is a single tree.



By permuting the leaf-nodes of [2], the leaf-nodes (1,3) and (2,3) can be made adjacent.

(a) A Transformation of a pq-tree



No allowable transformations of the pq-tree will make the leaf-node (1,7) adjacent to the leaf-node (3,7).

(b) G is nonplanar

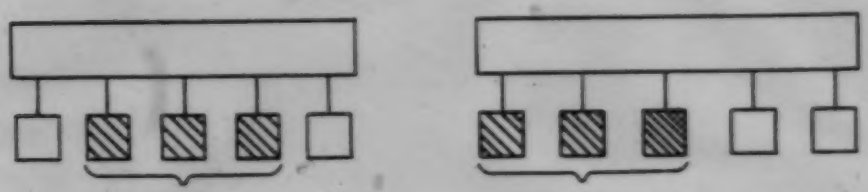
Figure 17. Leaf-nodes that can and cannot be made adjacent

Definition 3.2.8

The **minimal pertinent subtree**, P_k, of the pq-tree T_k is the smallest subtree of T_k such that P_k is made up of only pertinent nodes and all pertinent leaf-nodes of T_k are leaf-nodes of P_k.

On the k+1'st iteration of the embedding procedure the attempt to make the nodes of S_{k+1} adjacent on the periphery of T_k involves two passes through T_k. A first pass, a **bubble** pass, is performed to identify the minimal subtree P_k of T_k and, during the second pass, a **reduction** pass, the nodes of P_k are traversed and a modification operation is performed on each traversed node in an attempt to make all the pertinent leaf-nodes of T_k adjacent.

The bubble pass, for the k+1'st iteration, is performed by traversing nodes of T_k from the pertinent leaf-nodes up. Only nodes known to be in P_k are traversed during this pass. The traversed nodes are classified into two types, **blocked** nodes, and **unblocked** nodes. A node is classified as blocked if it is an interior child of a q-node and both of its adjacent siblings are blocked or have not been traversed. It is classified as unblocked otherwise (if its parent is a p-node, or it is an endmost child of a q-node, or it is an interior child of a q-node with an unblocked adjacent sibling) (see Figure 18). A blocked node becomes unblocked if one of its adjacent siblings becomes unblocked.



(a) Blocked Nodes (b) Unblocked Nodes

Figure 18. Blocked and Unblocked Nodes

Definition 3.2.9

A block of nodes is a maximum chain of adjacent blocked siblings.

A node must be identified as part of P_k before it can be traversed. All the pertinent leaf-nodes are part of P_k. A p-node or a q-node is part of P_k if it is the parent of an unblocked node. When the root of P_k is identified then the bubble pass halts.

Following the bubble pass, the nodes of T_k can be grouped as illustrated in Figure 19. The node rt in Figure 19 is the root of T_k. b1, b2, ..., bh are blocks of T_k, where $h \geq 0$. Let R_{bi} be the set of pertinent leaf-nodes whose ancestors are the blocked nodes in b_i, $i = 1, 2, \ldots, h$, and let $R_u = S_{k+1} - (R_{b1} \cup \ldots \cup R_{bh})$. R_u is the set of pertinent leaf-nodes whose ancestors are only unblocked nodes. If $|R_u| = 0$ and $h \geq 2$, or $|R_u| > 0$ and $h \geq 1$, then G is nonplanar and the embedding procedure halts. The presence of a block b_i, $i > 0$, indicates that there are pertinent

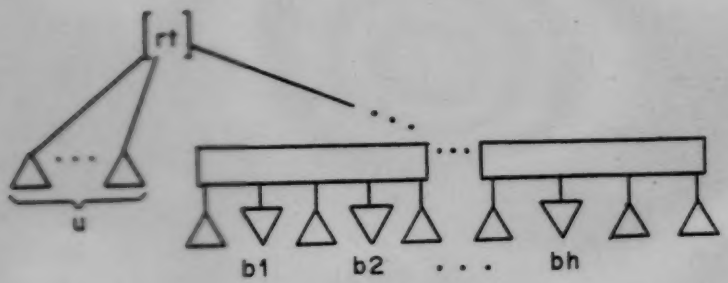


Figure 19. The Form of Tk after a bubble pass

leaf-nodes which are surrounded by nonpertinent leaf-nodes. If there exist pertinent leaf-nodes other than those in Rbi, then all the pertinent leaf-nodes cannot be made adjacent, so G is nonplanar. If h = 0, or h = 1 and |Ru| = 0, then the reduction pass is performed.

The reduction pass is an attempt to reduce Tk to a tree in which all the pertinent leaf-nodes are adjacent. Let m be the number of nodes in Pk. The reduction pass involves a bottom up traversal of Pk. A node is traversed only if all of its pertinent children have been traversed. The pertinent leaf-nodes are traversed first. Let xi, $1 \leq i \leq m$, be the i'th node traversed in Pk. The reduction of Tk is performed by creating a succession of trees Tk(0), Tk(1), Tk(2), Tk(3), ..., Tk(m), where Tk(0) = Tk and each tree Tk(i) is the tree formed by the modification operation M(Tk(i-1),xi) performed when xi is traversed, for $1 \leq i \leq m$.

Let Rxi be the set of pertinent leaf-nodes in the subtree below xi. The operation M(Tk(i-1),xi), while

maintaining the ordering restrictions x_j imposes on its children, permutes the leaf-nodes below x_j in an attempt to make the leaf-nodes of $R x_j$ adjacent. If no template pattern is found which accomplishes this, then the graph G is nonplanar. If all the nodes in P_k are visited and matched to a template pattern, then all the pertinent leaf-nodes of T_k are adjacent in $T_k(m)$. Figure 20 illustrates the transformations from T_3 to $T_3(5)$ for the graph K(3,3). The shaded square nodes in Figure 20 are pertinent leaf-nodes.

Definition 3.2.10

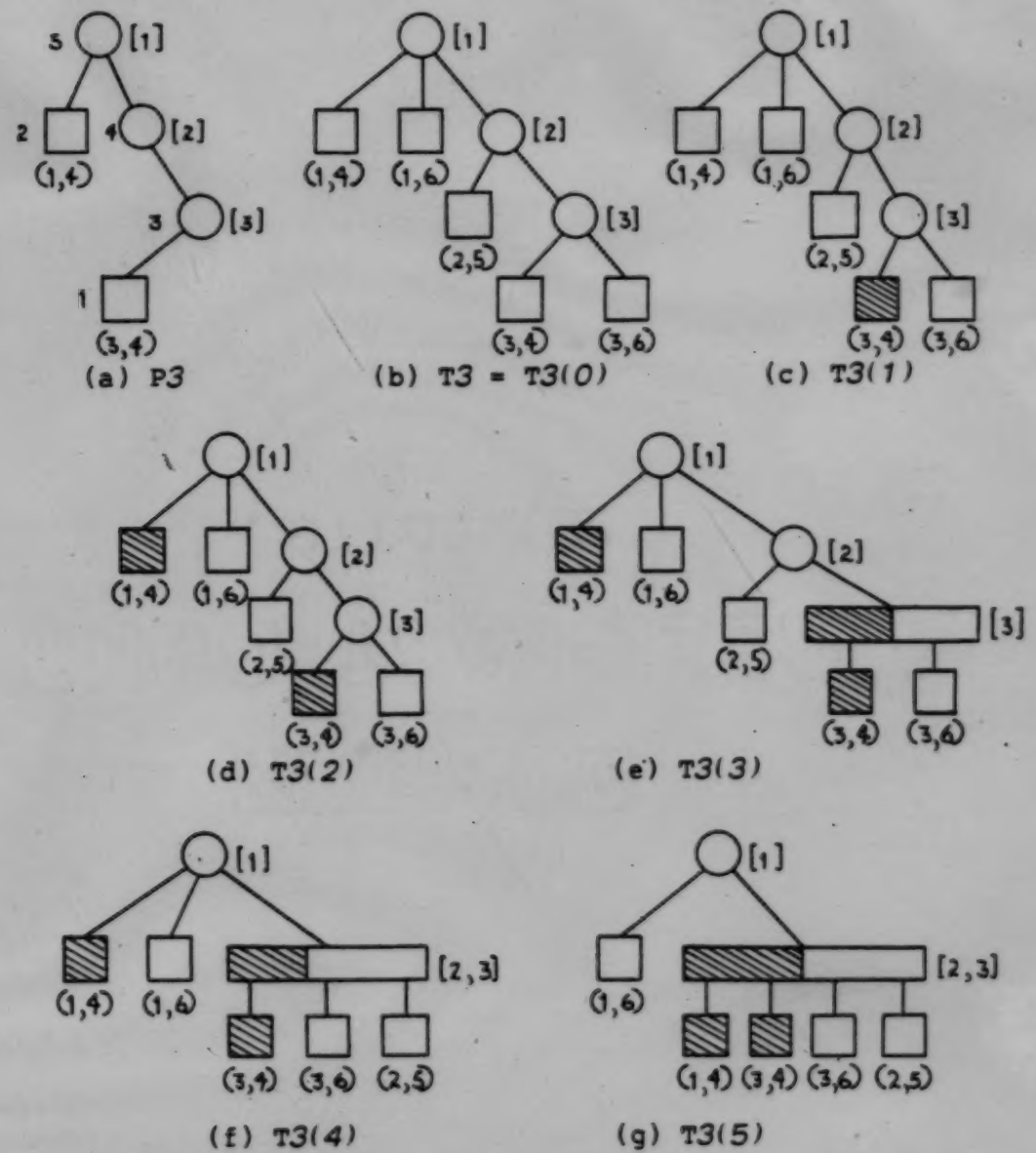
During the reduction pass, a node is labelled as **full** if either

a. the node is a pertinent leaf-node, or

b. all of the children of the node are full nodes.

In $T_k(m)$ either all the pertinent leaf-nodes are descendants of one full node, or all the pertinent leaf-nodes are descendants of adjacent full nodes having the same parent. The full nodes of $T_k(m)$ are replaced by a single p-node representing the real vertex numbered k+1. Then, leaf-nodes representing edges directed from vertex k+1 to a higher numbered vertex are made children of the new p-node. The resulting tree is T_{k+1}.

The complete embedding procedure is shown in Algorithm 3.2.1. The algorithm accepts as input a biconnected, st-numbered graph G and returns the value TRUE if G is planar, FALSE otherwise.



Leaf-nodes in S4 = {(3,4), (1,4)} must be made adjacent.

Figure 20. Transforming T3

Algorithm 3.2.1 (The pq-tree Embedding Procedure)

1. Form T_1 by creating a p-node representing vertex 1 of G. For each edge emanating from vertex 1 create a leaf-node to represent that edge and make the leaf-node a child of the p-node.
2. For k=1, 2, ..., n-2 do
   - 2.1 Set S_{k+1} = {all leaf-nodes representing edges with virtual vertex k+1}.

   Comment: Merge all virtual vertices labelled k+1 by making all pertinent leaf-nodes of T_k adjacent.
   - 2.2 Perform a bubble pass on T_k. Let h be the number of blocks in T_k, and let R_U be the set of pertinent leaf-nodes whose ancestors are all unblocked nodes.
   - 2.3 If (h ≥ 2) or ((h = 1) and ($|R_U| > 0$)), then return FALSE.

     Comment: The pertinent leaf-nodes of T_k cannot be made adjacent hence G is nonplanar.

   Comment: Otherwise, perform a reduction pass on T_k using P_k.
   - 2.4 Let m be the number of nodes in P_k.
   - 2.5 Set $T_k(0) = T_k$.
   - 2.6 For i = 1, 2, ..., m do
     - 2.6.1 let x_i be the i'th node of P_k traversed,

       Comment: Perform $M(T_k(i-1), x_i)$.
     - 2.6.2 compare each template pattern with x_i and its children until a match is found or all the patterns are exhausted,

2.6.3 if a match is found, then modify x_i and its children using the corresponding template replacement, creating $T_k(i)$,

2.6.4 else, return FALSE.

2.7 Replace all full nodes in $T_k(m)$ with a p-node representing the real vertex numbered k+1.

2.8 For each edge (k+1,j), where j > k+1, create a leaf-node representing the edge and make the leaf-node a child of the p-node created for the real vertex numbered k+1. The tree T_{k+1} is formed.

3. Return TRUE. **Comment:** G is planar.

Some minor bookkeeping operations have been omitted from the procedure which are worthy of mention. In particular, step 2.8 of Algorithm 3.2.1 includes a check that T_{k+1} is a **proper** pq-tree. A pq-tree is proper if the p-nodes and q-nodes of the tree have the properties of Definition 3.2.5. That is, any node that has only one child is deleted from the tree and any q-node with only two children is replaced by a p-node. These operations ensure that the trees T_k, for $1 \leq k \leq n-1$, are as compact as possible and that the difference between p-nodes and q-nodes is maintained.

Chapter 4

Edge Partitioning Algorithms

Edge partitioning algorithms separate the set of edges E of a graph G = (V, E) into k disjoint, nonempty subsets E(H$_i$), $1 \leq i \leq k$. The algorithms studied in this chapter produce a planar partition of G. Edge partitioning algorithms are examined because of their potential use as heuristic algorithms for determining a minimal planar partition of a graph.

The three algorithms described in this chapter are new. The algorithms are (1) a Spanning Tree Edge Partitioning (STEP) algorithm, (2) a Path Finding Edge Partitioning (PFEP) algorithm, and (3) a pq-tree Edge Partitioning (PQEP) algorithm.

Previous work on edge partitioning algorithms includes an algorithm from Fisher and Wing [Fis 66]. They present a planarity algorithm and discuss its modification to form an algorithm which identifies a planar subgraph of a graph. The planarity algorithm uses the incidence matrix of a graph as the main data structure. It operates on the same principle as the path finding algorithm described in section 3.1; it finds a cycle C in the graph G, decomposes G into C and the segments induced by C, and then determines the planarity of G by attempting to embed each segment in a plane.

The segment embedding proceeds as follows. First, the segments are partitioned into sets. One arbitrary segment is

placed in the set B0 while the other segments of C are placed in a set R of segments remaining to be partitioned. For $i \geq 1$, Bi is the set of all segments that interlace with B$i-1$ and Bi is formed as follows:

1. Bi is initialized to the empty set Ø.
2. Rtemp is initialized to Ø, where Rtemp is the set of segments that could not be placed in Bi.
3. For each segment S in R do
   - 3.1 If S interlaces with at least one segment in B$i-1$ and S does not interlace with any segment already in Bi, then place S in Bi.
   - 3.2 If S interlaces with a segment in B$i-1$ and S interlaces with a segment in Bi, then the partitioning procedure fails. Declare G nonplanar and halt.
   - 3.3 If S does not interlace with any segment in B$i-1$, then place S in Rtemp.
4. If Bi = Ø (no segment in R interlaces with any segment in B$i-1$), then start a new series B0′, B1′, ... with an arbitrary segment of R′ = Rtemp placed in B0′.
5. If Rtemp = Ø, then the partitioning procedure has succeeded in partitioning the segments of C.
6. Else, R = Rtemp and repeat the process from step 1 for $i = i + 1$.

It is interesting to note that each series (B0, B1, ...), (B0', B1', ...), and so on, is equivalent to a block in the path finding algorithm. In the embedding of G, the segment in B0 is embedded on the inside of the cycle C and the segments in B*i*, i ≥ 1, are embedded on the opposite side of C from the segments in B*i-1*.

If the partitioning procedure succeeds, then, for each segment S, the planarity of (S U C) is checked. If S comprises (1) a single edge or (2) multiple edges going to a single vertex not on C, then S U C is planar and the segment does not have to be decomposed [Fis 66]. If, on the other hand, S is made up of (3) a connected subgraph (more than one vertex) plus the edges attached to C, then the planarity of S U C must be checked. This is done by finding a cycle C' in S U C (C' ≠ C) and partitioning each of the segments in S U C induced by C'. The decomposition process stops when all the segments induced by a cycle are of type (1) or (2) above.

The modified algorithm, to find a planar subgraph of a nonplanar graph, proceeds as follows: If the partitioning procedure fails when trying to place a segment S into a set B*i*, then S has edges removed from it (edges incident with a vertex of C) one at a time, until the remaining segment S' no longer interlaces with a segment in B*i* and S' still interlaces with a segment in B*i-1*. Then S' is placed in B*i*. During the edge deletion process, if S' ceases to interlace with any segment in B*i-1*, then the deletion process stops

and S' is placed in Rtemp.

The test performed on the modified algorithm in [Fis 66] using the graph K9 does not indicate how well the algorithm performs. The modified algorithm is not deemed to be worth further attention because it operates on the same principle as the path finding algorithm described in section 3.1 which is a more efficient algorithm.

4.1 The Spanning Tree Edge Partitioning Algorithm

The Spanning Tree Edge Partitioning (STEP) algorithm produces a planar partition for a graph G = (V, E) as shown in Algorithm 4.1.1.

Algorithm 4.1.1 (STEP)

1. Set j = 0, V(H) = V, and E(H) = E.
2. While H is nonplanar do
   - 2.1 Set j = j + 1, and H*j* = (V, Ø).
   - 2.2 Set D = Ø.
   - **Comment:** Construct a planar subgraph H*j* of H.
   - 2.3 For each connected component C*i* of H,
     - i = 1, 2, ..., r, do
       - 2.3.1 Set F = a spanning tree of C*i*.
       - 2.3.2 Let S = {(v,w): (v,w) is an edge of C*i* - F}.
       - 2.3.3 While (S ≠ Ø) and (|E(F)| < 3|V(C*i*)| - 6) do
         - 2.3.3.1 Set S = S - {(v,w)} for some edge (v,w) in S.
         - 2.3.3.2 Set E(F) = E(F) U {(v,w)}.

2.3.3.3 If F is nonplanar, then E(F) = E(F) - {(v,w)} and D = D U {(v,w)}.

2.3.4 If S ≠ Ø, then D = D U S.

2.3.5 Set E(H*j*) = E(H*j*) U E(F)

2.4 Set E(H) = D.

3. Set j = j + 1, V(H*j*) = V, and E(H*j*) = E(H).

4. The set {H*1*, H*2*, ..., Hq} is a planar partition of G, where q = j.

The STEP algorithm creates a depth first spanning tree and determines the planarity of a graph using the corrected version of the path finding planarity algorithm from Chapter 3.

4.1.1 The Correctness of Algorithm 4.1.1 (STEP)

Theorem 4.1.1 Each subgraph H*j* in the partition {H*1*, H*2*, ..., Hq} constructed by Algorithm 4.1.1 (STEP) is planar.

Proof: For each subgraph F of a connected component C*i* of H, an edge is added to F only if the resulting subgraph is planar. Thus F is alway planar. Since each subgraph H*j* is the union of the planar subgraphs F of the connected components of H, each H*j* is planar, for j < q. Hq is obviously planar.

4.1.2 Analysis of Time Complexity

If G is planar then the time complexity of the STEP algorithm is equal to the time complexity of the algorithm used to determine the planarity of G, which is 0(n).

If G is nonplanar then the time complextiy of the STEP algorithm is as follows:

1. Constructing the first planar subgraph requires testing at most e-n+1 edges in step 2.3. For each edge tested, 0(n) time is required to determine the planarity of F. Thus the time complexity is at most 0(ne) to construct the first subgraph of G.
2. For each planar subgraph H*j* constructed (except the last when the graph is planar) the time complexity is 0(n*j**e*j*), where n*j* is the number of vertices and e*j* the number of edges in H*j*, and 0(n*j**e*j*) is 0(ne) in the worst case.
3. Since there are q subgraphs in the partition, and q is 0(n), an upper bound on the time complexity is $0(n^2e)$ (or $0(n^4)$ for e of size $0(n^2)$).

4.1.3 The STEP Algorithm Summary

The Spanning Tree Edge Partitioning (STEP) algorithm separates the set of edges of a graph G into disjoint subsets producing a planar partition of G. Each subgraph in the partition is formed by adding edges to an initial spanning tree (or spanning forest if the graph is not connected). Each subgraph is planar by Theorem 4.1.1. In

addition, each subgraph H*j* is maximally planar in the sense that any edge of H - H*j* added to H*j* would result in a nonplanar subgraph. An upper bound on the time complextiy of the STEP algorithm is $O(n^2e)$.

4.2 The Path Finding Edge Partitioning Algorithm

The Path Finding Edge Partitioning (PFEP) Algorithm, 4.2.1, produces a planar partition of a graph G = (V, E) using a modified version of the path finding algorithm described in section 3.1. The algorithm produces a planar subgraph of the set of edges remaining to be partitioned and this is done repeatedly until all the edges of G are partitioned.

Algorithm 4.2.1 (PFEP)

1. Set j = 0, V(H) = V and E(H) = E.
2. While E(H) ≠ Ø do
   - 2.1 Set j = j + 1 and H*j* = (V, Ø).

   Comment: Construct a planar subgraph H*j* of H.
   - 2.2 For each biconnected component B*i* of H, i = 1, 2, ..., r, do
     - 2.2.1 Input B*i* to a planar graph constructor that outputs a planar subgraph B*i*′ of B*i* (if B*i* is planar then B*i*′ = B*i*) and a set of edges D*i* = E(B*i*) - E(B*i*′).
     - 2.2.2 E(H*j*) = E(H*j*) U E(B*i*′).
   - 2.3 Set E(H) = D1 U D2 U ... U D*r*.

3. The set {H1, H2, ..., Hq} is a planar partition of G, where q = j.

4.2.1 The Planar Graph Constructor

The planar graph constructor in step 2.2.1 of Algorithm 4.2.1 is a modified version of the path finding planarity algorithm described in section 3.1. The modifications to the planarity algorithm cause edges to be deleted from the input graph whenever the graph is determined to be nonplanar.

Let G be the graph input to the planar graph constructor and let C be the initial cycle of G traversed by the modified embedding procedure. The modified procedure maintains, for each vertex in a block, a list of all the back edges represented by that vertex. A vertex x in a block represents the back edge (v,x) of a path p in a segment and also represents the back edges of all the special paths attached to p.

The modified procedure performs the first planarity test on a segment S of C in two parts. First, without actually shifting any previously embedded segments, it determines whether S is embeddable. Secondly, if S is embeddable, then it shifts segments and forms the block b for S. If S is not embeddable, then it deletes all the back edges from the segment so that the remaining edges of S can be embedded. Note that the deletion of edges from S during the planarity test of S U t is handled recursively and separate processing is not required.

If S U C is determined to be nonplanar during part(b) of the second planarity test, then back edges are again deleted from S. S U C is determined to be nonplanar whenever a block b' = [bx', bz'] is found which contains interlacing segments with vertices of attachment on t (see Figure 21). The back edges chosen for deletion from S are all those back edges from either:

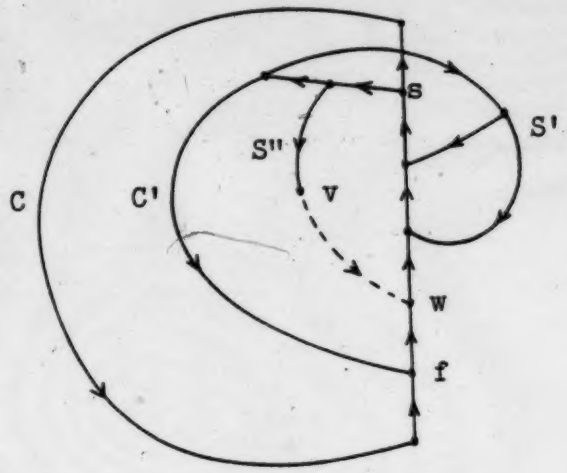
1. {(v,x): x is in bx' and x represents (v,x)}, or
2. {(v,z): z is in bz' and z represents (v,z)}.

The choice is made so that the fewest edges are deleted. After the back edges have been deleted from S the procedure continues with the planarity test of S U C.

The two algorithms of the modified embedding procedure are given below.

Algorithm 4.2.2 (The First Planarity Test)

1. Set INTERLACE = true, EMBEDDABLE = true, and let p: s -->* f be the leading path of S.
2. While (the top block b' = [bx', bz'] on B is not an end-of-stack marker) and EMBEDDABLE and INTERLACE do
   2.1 Let x be the largest vertex in bx' and let z be the largest vertex in bz'.
   Comment: Test whether S is embeddable with respect to the segments in the block b':
   2.2 If x > f and z > f, then EMBEDDABLE = false.
   Comment: S' and S" interlace with S, and S is not embeddable.



The back edge (v,w) will be deleted and S' shifted to the inside of C'.

Figure 21. Deleting a Back Edge

2.3 Else, if ($x > f$ and $z \leq f$) or
($x \leq f$ and $z > f$), then
remove the top block from B and save it.
Comment: There may still be more constraints on S.

2.4 Else, INTERLACE = false.
Comment: $x \leq f$ and $z \leq f$ so S does not interlace with either S' or S" and S is embeddable.

3. Restore the blocks deleted from B during step 2.
4. If (¬EMBEDDABLE), then delete all the back edges from S and embed all the tree edges of S.
5. Else, **Comment:** S is embeddable.
   5.1 Set INTERLACE = true, and b = [bx, bz] = [Ø, Ø]
   5.2 While (the top block b' = [bx', bz'] on B is not an

end-of-stack marker) and INTERLACE do

5.2.1 Let x be the largest vertex in bx' and let z be the largest vertex in bz'.

5.2.2 If x > f and z ≤ f, then

5.2.2.1 bx = bz' U bx,

5.2.2.2 bz = bx' U bz,

5.2.2.3 delete the top block from B.

5.2.3 Else, if x ≤ f and z > f, then

5.2.3.1 bx = bx' U bx,

5.2.3.2 bz = bz' U bz,

5.2.3.3 delete the top block from B.

5.2.4 Else, INTERLACE = false.

Comment: x ≤ f and z ≤ f and no more segments interlace with S.

Comment: p is embedded on the inside of C and the block b contains all vertices of attachment for all segments S' interacting with S.

5.3 If p is a normal path, then

5.3.1 bx = bx U {(f,p)}

5.3.2 BE(p) = {(v,f)}, where (v,f) is the back edge of p and BE(p) is the list of back edges represented by the vertex f.

5.4 Else, **Comment:** p is a special path.

5.4.1 let p' be the normal path to which p is attached,

5.4.2 BE(p') = BE(p') U {(v,f)}, where (v,f) is the back edge of p.

5.5 If S is a nontrivial segment, then

5.5.1 place b on the block stack,

5.5.2 place an end-of-stack marker on top of b on the stack.

5.6 Else, **Comment:** S is a trivial segment.

if b ≠ [Ø, Ø], then make b the top block on B.

6. Return TRUE.

Algorithm 4.2.3 (Part (b) of the Second Planarity Test)

1. Set b = [bx, bz] = [Ø, Ø], and d = Ø, where d will be the set of edges deleted from S.
2. While the top block b' = [bx', bz'] on B is not an end-of-stack marker do

2.1 Let x' be the smallest vertex of bx' and let z' be the smallest vertex of bz'.

2.2 If x' > f and z' > f, then

Comment: S U C is nonplanar.

2.2.1 Let dx' be the set of back edges represented by the vertices in bx' and let dz' be the set of back edges represented by the vertices in bz'.

2.2.2 If |dx'| < |dz'|, then

2.2.2.1 bx' = Ø.

2.2.2.2 d = d U dx'.

2.2.3 Else,

2.2.3.1 bz' = Ø.

2.2.3.2 d = d U dz'.

2.3 Else, if x' > f and z' ≤ f, then

2.3.1 bx = bx' U bx (note that bz' = Ø in this case),

2.3.2 delete b' from B.

2.4 Else, if x' ≤ f and z' > f, then

2.4.1 bx = bz' U bx (note that bx' = Ø in this case),

2.4.2 delete b' from B.

3. Delete the end-of-stack marker from B.

Comment: Combine b with the top block b' of B (the block formed when the leading path p of S was embedded).

4. Set bx = bx' U bx.

5. Set bz = bz'.

6. Delete b' from B.

7. If b ≠ [Ø, Ø] then make b the top block on B.

8. Return TRUE.

4.2.2 The Correctness of Algorithm 4.2.1 (PFEP)

In the following discussion let G be the graph input to the planar graph constructor, let C be the initial cycle of G traversed by the modified embedding procedure, and let C have m segments $S1$, $S2$, ..., Sm, where Sk is the k'th segment of C tested for embeddability. Let dk be the set of edges deleted from Sk, let Sk' be the portion of Sk embedded, and let Gk be the portion of G embedded after the first k segments of C are tested, that is, $G0$ = C, $G1$ = $S1'$ U C = $G0$ U $S1'$, $G2$ = $G1$ U $S2'$, ..., Gm = $Gm-1$ U Sm'. The

subgraph G*m* is the graph output by the planar graph constructor in step 2.2.1 of PFEP.

Lemma 4.2.1 If the segment S*i*, 1 ≤ i ≤ m, interlaces with a previously embedded segment S*j'*, j < i, then Algorithm 4.2.2 will detect the interlace.

Proof: Let S*i* be the segment which is to embedded and let s*i* be its start vertex. The blocks of B checked in Algorithm 4.2.2 only contain vertices of attachment w representing back edges still in the graph such that w < s*i*. Therefore, Theorem 3.1.3 still applies to S*i* so the algorithm can correctly determine if S*i* interlaces with any previously embedded segments.

Lemma 4.2.2 If two segments S*i'* and S*j'* of C interlace, j < i, and S*i'* is embedded on the inside of C, then there is a block b = [bx, bz] on B that represents the interaction, and S*j'* is represented in bz.

Proof: Since j < i, S*j'* is embedded first, and is represented in some block b' on B. After the leading path p: s -->* f of S*i* is generated Algorithm 4.2.2 is performed. By Lemma 4.2.1 above, all segments S*j'* that interlace with S*i* are determined. If S*i'* and S*j'* interlace, then S*i'* U t is not a tree. Therefore, S*i* must have passed the first planarity test with p embedded on the inside of C and each segment S*j'* interlacing with S*i* embedded on the outside of C and represented in a single block b. The segment S*i'*

resulting from the deletion of edges from S*i* will not interlace with any segment that did not interlace with S*i*. Therefore, the block b (or some block containing b) represents the interaction between S*i'* and the segments with which it interlaces and if S*i'* is embedded on the inside of C, then any interlacing segments are embedded on the outside of C.

Lemma 4.2.3 If a segment S*i*, $1 \leq i \leq m$, passes the first planarity test, then S*i* can be embedded so that it does not overlap with any segment S*j'* of C, $j < i$, and so that the segments S*j'* do not overlap.

Proof: If any segment S*i* interlaces with two segments that interlace with each other, then from Lemma 4.2.1 and Lemma 4.2.2, it follows that the interaction will be detected. Therefore, if S*i* passes the first planarity test, then S*i* can be embedded with respect to the segments previously embedded.

Lemma 4.2.4 Given a nontrivial segment S, assume that the subgraph $S^{\circ} \cup t$ (S° a subgraph of S) returned from the recursive call to the embedding procedure is planar. If d is the set of back edges deleted from S° in Algorithm 4.2.3, then $(S^{\circ} - d) \cup C$ is planar.

Proof: For each block b' = [bx', bz'] formed during the recursive call to embed S°, either C - t can be embedded with the segments represented in b' or C - t interlaces with at least one segment S' represented in bx' and at

least one segment S" represented in bz'. Removing all the back edges represented in bx' or bz' means that C - t no longer interlaces with the affected segments and that the segments remaining in b' that interlace with C - t do not interlace with each other. Since Algorithm 4.2.3 operates on all the blocks formed during the recursive call, C - t can be embedded with respect to all the segments in (S° - d) U t, which means that (S° - d) U C is planar.

Theorem 4.2.5 For any graph G input to the planar graph constructor, the subgraph G*m* output by the constructor is planar.

Proof: To show that G*m* is planar it is shown that each subgraph G*i*, $0 \leq i \leq m$, is planar.

I For i = 0, G*0* = C which is planar.

II Assume that for all $i < k$, G*i* is planar.

III Consider G*k*. G*k-1* is planar by II. To show that G*k* is planar, we must show that the two planarity tests hold for S*k'*. There are two cases to consider, either d*k* = Ø, or d*k* ≠ Ø.

(1) If d*k* = Ø, then S*k* passed both planarity tests so it follows from Lemma 4.2.3 and Lemma 4.2.4 that G*k* is planar.

(2) If d*k* ≠ Ø, then there are a number of subcases to consider:

(i) If edges are deleted from S*k* during the

first planarity test, then all the back edges of S_k are deleted leaving a segment S_k' which is composed of all the tree edges of S_k. Since $S_k' \cup t$ is a tree, $G_k = G_{k-1} \cup S_k'$ is planar. Since S_k' does not interlace with any previously embedded segments and will not interlace with any segments yet to be embedded, the blocks currently on the block stack are correct and no new block is required.

(ii) Testing the planarity of $S_k \cup t$ requires a recursive call to the modified embedding procedure with $S_k \cup t$ as input. Assume for the moment that the subgraph $S_k^\circ \cup t$ (the subgraph returned from the recursive call) is planar.

(iii) When the planarity of $(S_k^\circ \cup t) \cup (C - t)$ is tested, let d be the set of back edges deleted from S_k°, then, by Lemma 4.2.4, $(S_k^\circ - d \cup C)$ is planar. Let $S_k' = S_k^\circ - d$. Since Lemma 4.2.3 holds for S_k and S_k', it follows that S_k' passes the two planarity tests so G_k is planar.

For part (ii), notice that at the lowest level of recursion, all the segments are trivial and only (i) or (iii) occurs, hence the subgraph returned from the lowest recursive call is planar and part

(ii) follows. Therefore, G*k* is planar.

From the preceding, it follows that G*i* is planar, for $0 \leq i \leq m$. Therefore, the subgraph G*m* output from the planar graph constructor in step 2.2.1 of Algorithm 4.2.1 is planar.

Theorem 4.2.6 Each subgraph H*j* of the partition {H*1*, H*2*, ..., Hq} constructed by Algorithm 4.2.1 (PFEP) is planar.

Proof: Each subgraph B*i'* constructed in step 2.2.1 of Algorithm 4.2.1 is planar by Theorem 4.2.5. Since any subgraph H*j*, of the partition, $1 \leq j \leq q$, is the union of those planar subgraphs B*i'* of the biconnected components B*i* of the graph H, $1 \leq i \leq r$, it follows that H*j* is planar.

Unfortunately, a segment S*i* may be embeddable but be declared unembeddable. If S*i* is declared unembeddable, then it interlaces with two segments S*j'* and S*k'*, $j < k < i$, represented in the same block and embedded on opposite sides of C. In the original embedding procedure, this meant that S*j'* and S*k'* interlaced. In the modified procedure, although S*j'* interlaces with S*k*, S*j'* and S*k'* may not interlace thus S*i* may in fact be embeddable. Fortunately, the problem only arises if (1) S*k* has the same start and finish vertices as S*j* and (2) S*k'* only has two vertices of attachment. Since S*j'* interlaces with S*k*, they both must have three vertices of attachment. Since S*k'* only has two vertices of

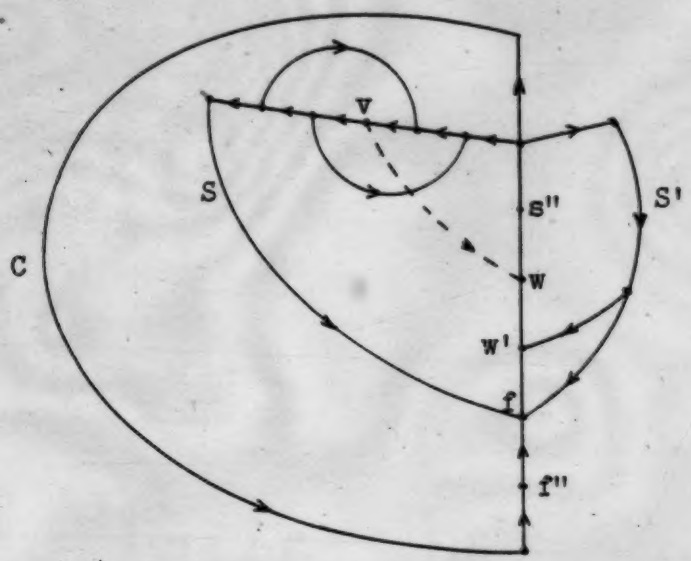
attachment, Lemma 3.1.2 no longer applies and Sk' does not interlace with Sj'. However, the two segments are still represented in a single block as if they interlaced (see Figure 22).

This incorrect representation can only occur between segments that have the same start and finish vertices because of the following lemma.

Lemma 4.2.7 For any segment S*i*, 1 ≤ i ≤ m, let f be its finish vertex. Either S*i*' has f as its finish vertex or S*i*' has only one vertex of attachment (there are no back edges in S*i*').

Proof: Given any segment S*i* with leading path p: s -->* f, if S*i* passes the first planarity test, then a block b is formed representing the interactions between S*i* and segments S*j*', j < i, and p is embedded. When Algorithm 4.2.3 is performed for S*i*, the back edge of p is not involved in the testing as it is not part of a segment in S*i* U t*i*. Therefore, the back edge of p is not deleted and f remains as a vertex of attachment of S*i*'. If S*i* fails the first planarity test, then S*i*' has no back edges and therefore, s is the only vertex of attachment of S*i*'.

Note that the back edge of p may be deleted at a higher level in the recursion. However, at that point, S*i* has ceased to be a segment.



If the back edge (v,w) is deleted, then S and S' no longer interlace.

Figure 22. Loss of Interaction

4.2.3 Analysis of Time Complexity

The modified embedding procedure examines each edge of G once, and performs only a linear number of block stack operations. Therefore, the modified procedure has a time complexity of $0(e)$. Since the number of subgraphs in the planar partition is $0(n)$, the time complexity of Algorithm 4.2.1 is $0(ne)$ (or $0(n^3)$ for e of size $0(n^2)$).

4.2.4 The PFEP Algorithm Summary

The Path Finding Edge Partitioning (PFEP) algorithm constructs a planar partition of a graph G. Each subgraph in the partition is formed using a modified version of the path

finding planarity algorithm from section 3.1. The algorithm accepts a biconnected graph as input and deletes edges from it until the remaining subgraph is planar. The only modifications made to the algorithm, as described in section 3.1, are modifications that cause the deletion of edges from the region of the graph being embedded when the graph is declared nonplanar. If the graph is declared nonplanar during the embedding of some segment S_i, then the only edges considered for deletion are edges from S_i. No backtracking (deletion of edges from segments already embedded) or lookahead (examining the ramifications of deleting a particular edge) is employed in deciding which edges to delete.

The modifications are designed to minimize the extra processing time required while deleting as few edges as possible from the graph. The subgraphs produced by the planar graph constructor may not be maximally planar for two reasons. First, a graph could be declared nonplanar when it is in fact planar, as explained in section 4.2.2. Second, the deletion of edges in Algorithm 4.2.3 from a segment S_i may make unnecessary the deletions made during the planarity test of $S_i \cup t_i$.

4.3 The pq-tree Edge Partition Algorithm

The pq-tree Edge Partitioning (PQEP) Algorithm, 4.3.1, produces a planar partition of a graph G = (V, E) using a modified version of the pq-tree planarity algorithm. The pq-tree algorithm determines whether or not a graph is planar. Its modified version produces a planar subgraph of a nonplanar graph.

Algorithm 4.3.1 (PQEP)

1. Set j = 0, V(H) = V, and E(H) = E.
2. While E(H) ≠ Ø do
   - 2.1 Set j = j + 1 and H*j* = (V, Ø)
     Comment: Construct a planar subgraph H*j* of H.
   - 2.2 For each biconnected component B*i* of H, i = 1, 2, ..., r, do
     - 2.2.1 Input B*i* to a planar graph constructor that outputs a planar subgraph B*i'* of B*i* (if B*i* is planar then B*i'* = B*i*) and a set of edges D*i* = E(B*i*) - E(B*i'*).
     - 2.2.2 E(H*j*) = E(H*j*) U E(B*i'*).
   - 2.3 Set E(H) = D*1* U D*2* U ... U D*r*.
3. The set {H*1*, H*2*, ..., Hq} is a planar partition of G, where q = j.

4.3.1 The Planar Graph Constructor

The planar graph constructor used to produce a subgraph B_i' of a biconnected component B_i of H, $1 \leq i \leq r$ in step 2.2.1 of Algorithm 4.3.1 is a modified version of the pq-tree planarity algorithm from Booth and Lueker [Boo 76] described in section 3.2. The embedding procedure was modified so that a planar subgrah is produced if the input graph is nonplanar. The modifications cause the deletion of edges from the input graph whenever the graph is nonplanar.

Let G be the graph input to the planar graph constructor, let n be the number of vertices of G, and let G_n be the subgraph of G output by the constructor. If G is declared nonplanar during the k+1'st iteration of the embedding procedure, then the edges chosen for deletion are chosen from the edges represented in S_{k+1} (edges with one endpoint numbered k+1) such that the remaining edges represented in S_{k+1} can be merged and the virtual edges in the resulting planar embedding lie in the same (outer) face. The deletion of edges can occur in the modified procedure following the bubble pass and/or during the reduction pass. A nonplanar declaration following the bubble pass of the k+1'st iteration, means that there are two or more sets of pertinent nodes separated by nonpertinent nodes in the pq-tree T_k. For the embedding process to continue only one group can be allowed to remain. Let the sets of pertinent nodes be u, b_1, b_2, ..., b_h, where u is the set of nodes that have no blocked nodes as ancestors and b_i, $1 \leq i \leq h$,

is a block (see Figure 23). The group of pertinent nodes chosen to remain is the one with the largest number of descendant pertinent leaf-nodes. Let R*u* be the set of pertinent leaf-nodes which are descendants of the nodes in u, and let R*bi* be the similar set for b*i*, 1 ≤ i ≤ h. The set of pertinent nodes chosen to remain in T*k* is the set x such that

$$|Rx| = \max\{|Ru|, |Rb1|, \ldots, |Rbh|\}.$$

All the pertinent leaf-nodes not in R*x* are deleted from T*k* and the resulting tree is made proper (to maintain the differences between p-nodes and q-nodes) forming a new tree T*k*. In addition, the elements in S*k+1* are replaced by the elements in R*x*. Then a bubble pass is performed on the new T*k* obtaining the mininal pertinent subtree P*k*.

A nonplanar declaration during the reduction pass of the k+1'st iteration means that a node and its children cannot be matched to any template pattern. In this case, the modified embedding procedure deletes pertinent leaf-nodes from the pq-tree until a template pattern matches the node and and its modified children. The minimum number of pertinent leaf-nodes are deleted from the tree. Let P*k* have m nodes and let x*i* (the i'th node traversed during the pass) be the node for which M(T*k*(*i-1*), x*i*) fails. The nodes x*j* of P*k* are labelled during the second step of M(T*k*(*j-1*), x*j*), 1 ≤ j ≤ m. They are labelled as full, singly partial, or doubly partial depending on the arrangement and labels of their children.

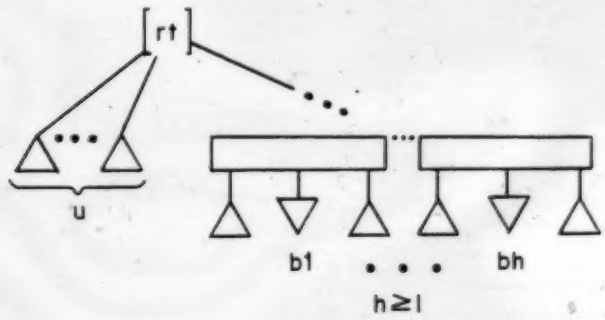


Figure 23. *Tk* nonplanar after a bubble pass

Definition 4.3.1

A node is labelled as **partial** if it is a q-node having full and empty children.

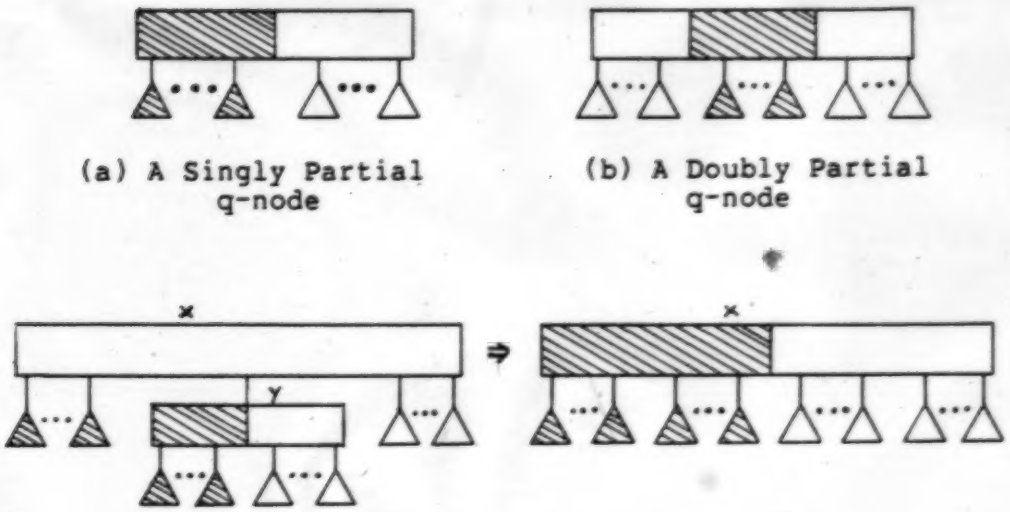
Definition 4.3.2

A **singly**, partial node is a partial node whose full children are adjacent and positioned at one end of the node (as in Figure 24(a)).

Definition 4.3.3

A **doubly**, partial node is a partial node whose full children are adjacent and surrounded by empty children (as in Figure 24(b)). For a node to be labelled doubly partial it must be the root of the pertinent subtree.

Note that if a q-node has a partial child, then that child is merged with the q-node during the template replacement process before the node is labelled (as shown in



(a) A Singly Partial q-node

(b) A Doubly Partial q-node

(c) Merging a Partial Child

Figure 24. The Form of q-nodes in a pq-tree

Figure 24(c)).

The node x*i* for which M(T*k*(*i*-1), x*i*) fails is either a p-node or a q-node since a leaf-node is always matched to a template pattern.

(1) If x*i* is a p-node, then T*k*(*i*-1) has the form shown in Figure 25, where x*m* is the root of P*k* and R*t* is the set of pertinent leaf-nodes that are not descendants of x*i*. R*t* is empty if i = m and nonempty otherwise. The nodes px1, px2, ..., pxh, where h ≥ 2, are the partial children of x*i*. Let R*px1*, R*px2*, ..., R*pxh* be the sets of pertinent leaf-nodes that are descendants of the respective partial nodes.

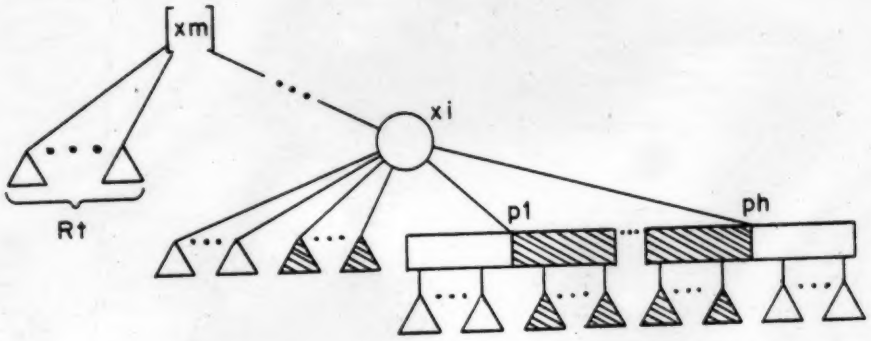


Figure 25. The Form of an Unmatched p-node

Pertinent leaf-nodes are deleted from $Tk(i-1)$ forming the tree $Tk(i-1)'$, such that $M(Tk(i-1)', xi)$ is successful. The deletion of pertinent leaf-nodes modifies the children of xi and/or makes xi the root of Pk. The steps involved are shown in Algorithm 4.3.2, where $rk+1$ is the set of pertinent leaf-nodes deleted during the k+1'st reduction pass.

Algorithm 4.3.2

1. Determine the partial children p1 and p2 of xi such that $Rp1$ is the largest and $Rp2$ is the second largest of $Rpx1$, $Rpx2$, ..., $Rpxh$.
2. Set ld = {pertinent leaf-nodes that are descendants of all the partial children of xi except p1 and p2}.
3. Delete all the elements of ld from $Tk(i-1)$.
4. Set $Sk+1 = Sk+1 - ld$.
5. Set $rk+1 = rk+1 \cup ld$.
6. If $(|Rt| \leq |Rp1|)$ and $(|Rt| \leq |Rp2|)$, then

6.1 Delete the elements of R_t from $T_k(i-1)$ forming $T_k(i-1)'$, and make x_i the root of P_k.

6.2 $S_{k+1} = S_{k+1} - R_t$.

6.3 $r_{k+1} = r_{k+1} \cup R_t$.

7. Else, if $|R_{p1}| \leq |R_{p2}|$, then

7.1 Delete the elements of R_{p1} from $T_k(i-1)$ forming $T_k(i-1)'$.

7.2 $S_{k+1} = S_{k+1} - R_{p1}$.

7.3 $r_{k+1} = r_{k+1} \cup R_{p1}$.

8. Else,

8.1 Delete R_{p2} from $T_k(i-1)$ forming $T_k(i-1)'$.

8.2 $S_{k+1} = S_{k+1} - R_{p2}$.

8.3 $r_{k+1} = r_{k+1} \cup R_{p2}$.

9. Perform $M(T_k(i-1)', x_i)$ forming $T_k(i)$.

(2) If x_i is a q-node, then the arrangement of the children of x_i is similar to that shown in Figure 26, where x_m is the root of P_k and R_t is the set of pertinent leaf-nodes that are not descendants of x_i. R_t is empty if $i = m$ and nonempty otherwise.

Definition 4.3.4

A **q-group** is a maximal group of adjacent children of a q-node such that all the nodes in the group are pertinent.

The node x_i has certain characteristics. First, x_i has at least one empty child or one partial child,

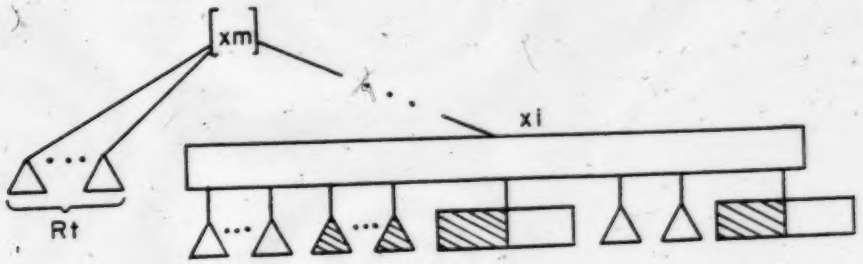


Figure 26. A Possible Configuration for an Unmatched q-node

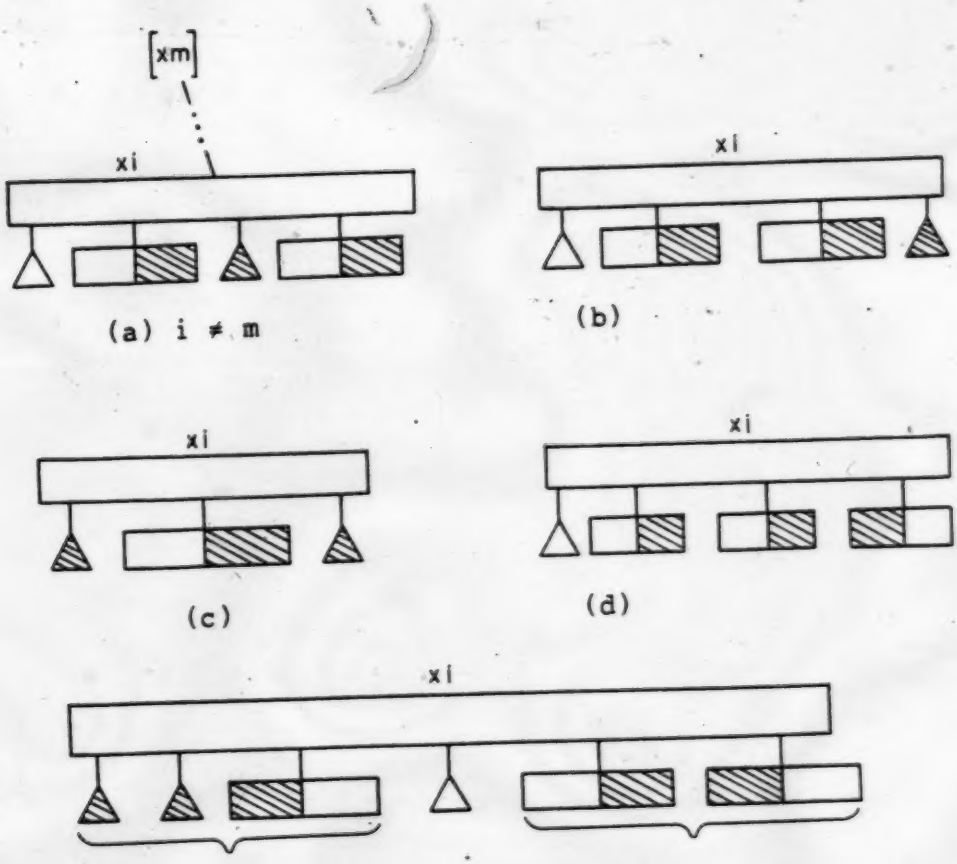


Figure 27. Possible Configurations of the Children of x*i*

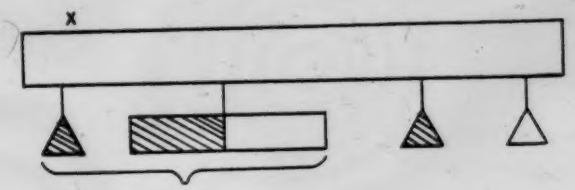
second at most one end child is empty, and third x_i has at most two q-groups. The first characteristic of x_i is a result of the failure of $M(T_k(i-1), x_i)$ and the second and third characteristics are a direct result of the bubble pass. Figure 27 shows some examples of the arrangement of the children of x_i.

Definition 4.3.5

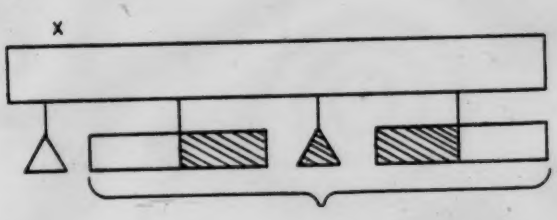
A **qsp-group** is a maximal group of adjacent, pertinent children of a q-node such that if it were the only group of pertinent children, then the node would be matched and labelled singly partial. A qsp-group of a node x consists of the pertinent children of x starting from one endmost full or partial child up to the first partial child or first empty child and including that child if it is partial (see Figure 28(a)).

Definition 4.3.6

A **qdp-group** is a maximal group of adjacent, pertinent children of a q-node such that if it were the only group of pertinent children, then the node would be matched and labelled doubly partial. A qdp-group of a node x consists of the pertinent children of x starting from the first full child after an empty child or from a partial child up to the next partial child or or empty child and including that child if it is partial (see Figure 28(b)).



(a) A qsp-group of a q-node x



(b) A qdp-group of a q-node x

Figure 28. Matchable Pertinent Groups of a q-node

The steps involved in modifying the children of x*i* to form a proper tree T*k*(*i*-1)′ such that M(T*k*(*i*-1)′, x*i*) succeeds and in the process deleting the minimum number of pertinent leaf-nodes are shown in Algorithm 4.3.3. Let r*k*+1 be the set of pertinent leaf-nodes deleted during the k+1'st reduction pass.

Algorithm 4.3.3

1. Find the qdp-group of x*i* with the largest number of pertinent leaf-nodes as descendants. Let R*dp* be the set of pertinent leaf-nodes of this group.
2. Find the qsp-group of x*i* with the largest number of pertinent leaf-nodes as descendants. Let R*sp* be the

set of pertinent leaf-nodes of this group.

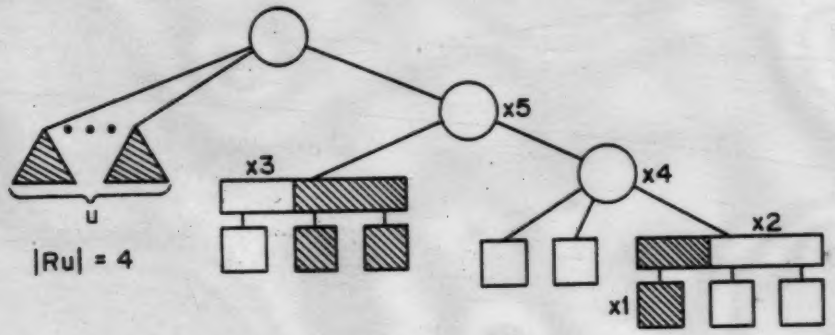
3. If $|Rt| + |Rsp| \leq |Rdp|$, then
   - 3.1 Delete all the pertinent leaf-nodes not in Rdp from $Tk(i-1)$ forming $Tk(i-1)'$, and make xi the root of Pk.
   - 3.2 $rk+1 = rk+1 \cup (Sk+1 - Rdp)$.
   - 3.3 $Sk+1 = Rdp$.
4. Else,
   - 4.1 Delete all the pertinent leaf-nodes not in $(Rsp \cup Rt)$ from $Tk(i-1)$ forming $Tk(i-1)'$.
   - 4.2 $rk+1 = rk+1 \cup (Sk+1 - (Rsp \cup Rt))$.
   - 4.3 $Sk+1 = Rsp \cup Rt$.
5. Perform $M(Tk(i-1)', xi)$ forming $Tk(i)$.

Note that the deletion of pertinent leaf-nodes during the k+1'st reduction pass does not require changing the minimal pertinent subtree Pk. Pk is used only to determine which nodes are traversed and their traversal order. Either xi becomes the root of Pk and no more nodes are traversed or the only nodes that would be deleted from Pk are children of xi and they have already been traversed. Thus the nodes remaining to be traversed and their traversal order is not changed by any deletions made during the reduction pass.

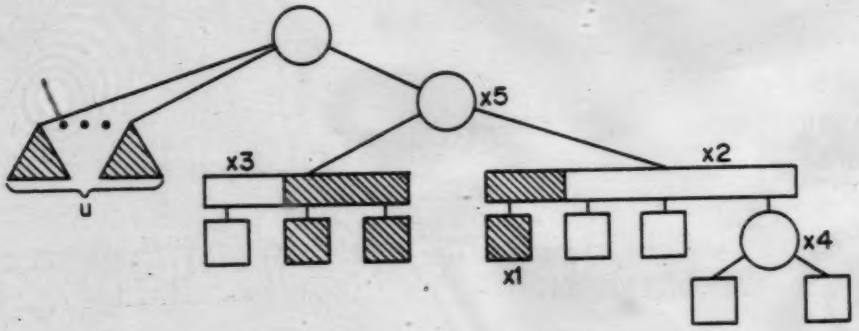
The deletion of pertinent leaf-nodes from $Tk(i-1)$, $1 \leq i \leq m-1$, does not require restarting the reduction pass. The pass can be allowed to run to completion and any ensuing deletion of pertinent leaf-nodes is valid, that is, the

deletions would have occurred even if the pass were restarted after each set of deletions. If M($T_k(i-1)$, x_i) fails, and i = m, or x_i is made the root of P_k, then the reduction pass is finished after M($T_k(i-1)'$, x_i) is performed so there are no further deletions and the reduction pass is finished. Otherwise, all the pertinent leaf-nodes deleted from $T_k(i-1)$ are descendants of x_i. If pertinent leaf-nodes are deleted from a child y of x_i, then all pertinent leaf-nodes are deleted from y making y an empty node. When x_i is matched to a template pattern, it is labelled as partial or is replaced by a node which is labelled as partial. The template matching of the parent of x_i (if any) depends only on the label of x_i and its position relative to its siblings. The deletion of descendant nodes of x_i does not affect its position relative to its siblings. If the reduction pass were restarted, then x_i (or the node replacing it after the modification operation) would still be labelled as partial because the pertinent children of x_i would have the same labels and relative positions in the pq-tree as they have in $T_k(i-1)'$. Since the deletion of descendant nodes of x_i does not affect the matching of x_i's parent, future matchings and deletions, performed during the reduction pass, are valid.

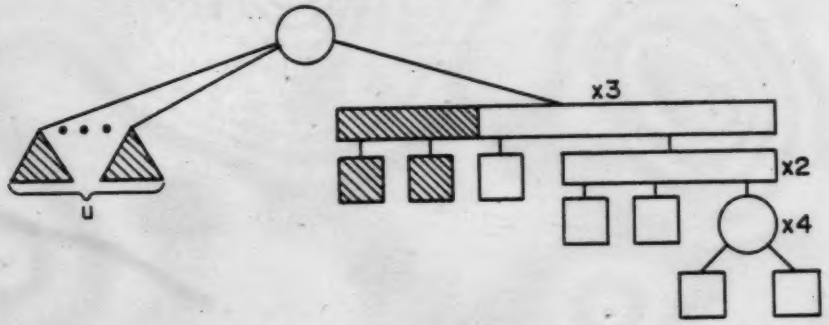
Unfortunately, the deletion of pertinent leaf-nodes affects template operations that occurred previously in the reduction pass and hence affects the form of T_{k+1} (see Figure 29). If pertinent leaf-nodes are deleted from the



(a) $Tk(i)$ before x4 is visited

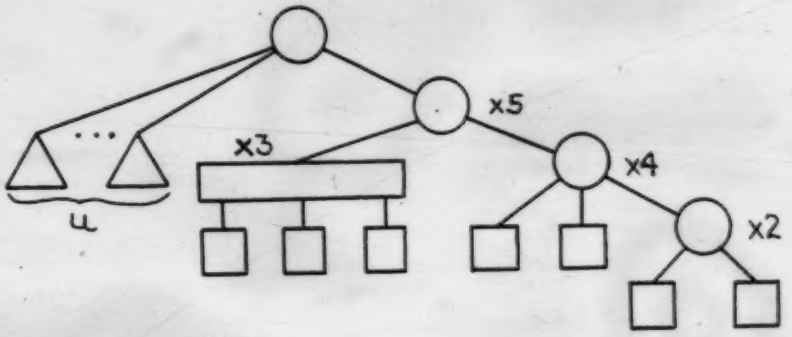


(b) After the reduction of x4: x5 is visited requiring the deletion of x1

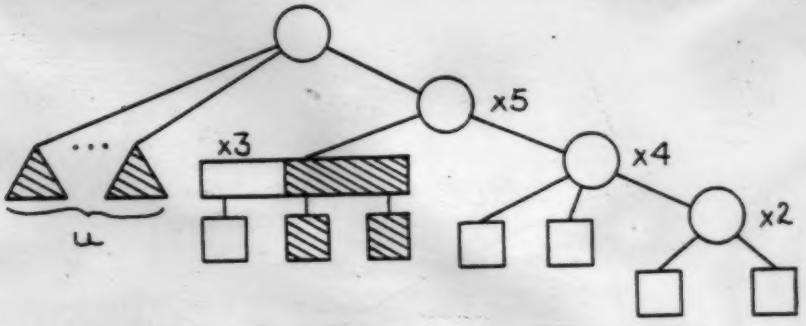


(c) After the deletion of x1 and the reduction of x5

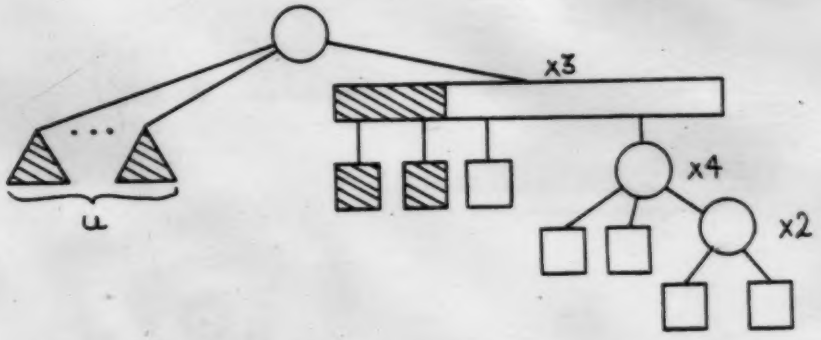
Figure 29. Tk before and after deletion of leaf-nodes during the Reduction Pass



(d) Tk' after x1 is deleted from Tk



(e) The tree prior to x5 being visited



(f) After the reduction of x5

Figure 29. Tk before and after deletion of leaf-nodes during the Reduction Pass

subtree rooted at a node y, then template operations that had been performed on that subtree are no longer necessary and any modifications made to that subtree which resulted in a change in the order of the remaining leaf-nodes or a change in the transformations allowed on those nodes should be undone. Since we really want the pq-tree for the remaining set of pertinent leaf-nodes, the reduction pass must be performed again. Let T_k' be the tree resulting from deleting the nodes of r_{k+1} from T_k such that T_k' is a proper pq-tree and let P_k' be the minimal pertinent subtree with m' nodes that results from the deletion from P_k of all nodes that are no longer pertinent. Then the second reduction pass of the k+1'st iteration is performed on T_k' using P_k'.

Algorithm 4.3.4 outlines the steps performed in the modified embedding procedure to obtain a planar subgraph G_n of the st-numbered, biconnected input graph G with n vertices. The set d_{k+1} is the set of pertinent leaf-nodes deleted during the k+1'st iteration and r_{k+1} is the set of pertinent leaf-nodes deleted during the first reduction pass of the k+1'st iteration.

Algorithm 4.3.4

1. Form T_1 by creating a p-node representing the vertex of G with st-number 1. For each edge emanating from vertex 1 create a leaf-node to represent that edge and make the leaf-node a child of the p-node.
2. For k=1, 2, ..., n-2 do

2.1 Set S_{k+1} = {all leaf-nodes representing edges with virtual vertex k+1}.

Comment: Merge all virtual vertices labelled k+1 by making all pertinent leaf-nodes of T_k adjacent.

2.2 Perform a bubble pass on T_k. Let h be the number of blocks in T_k and let R_U be the set of pertinent leaf-nodes whose ancestors are all unblocked nodes.

2.3 If ($h \geq 2$) or (($h = 1$) and $|R_U| > 0$), then

Comment: The pertinent leaf-nodes of T_k cannot be made adjacent.

2.3.1 find the set R_x for which $|R_x| = \max(|R_U|, |R_{b1}|, \ldots, |R_{bh}|)$,

2.3.2 delete all pertinent leaf-nodes from S_{k+1} and T_k not in R_x (delete the corresponding edges from G) forming a new tree T_k,

2.3.3 set $d_{k+1} = S_{k+1} - R_x$,

2.3.4 set $S_{k+1} = R_x$,

2.3.5 perform a bubble pass on T_k.

2.4 Form a temporary copy C_k of the portion of T_k required during the reduction pass.

2.5 Perform a reduction pass on C_k using P_k and obtain r_{k+1}.

2.6 Remove the leaf-nodes in r_{k+1} from T_k forming T_k' and a modified minimal pertinent subtree P_k' with m' nodes.

2.7 Set $d_{k+1} = d_{k+1} \cup r_{k+1}$.

2.8 Perform a reduction pass on T_k' using P_k' and

return $T_{k'}(m')$.

2.9 Replace all full nodes in $T_{k'}(m')$ with a p-node representing the real vertex numbered k+1.

2.10 For each edge (k+1,j), where j > k+1, create a leaf-node representing the edge and make the leaf-node a child of the p-node created in step 2.9. The tree T_{k+1} is formed (the corresponding subgraph G_{k+1}, with the edges of G_k and the edges corresponding to the leaf-nodes of $S_{k+1} - r_{k+1}$, is embedded in the plane and has all the edges of F_{k+1} in its outer face).

Comment: The subgraph G_{n-1} corresponding to T_{n-1} is embedded in the plane and all the edges of F_n have higher endpoint n and are in the outer face of the embedding.

3. Merge the virtual vertices numbered n in the embedding, creating the real vertex n. G_n is the resulting graph.

4. Set D = {edges represented by $d_1 \cup d_2 \cup \ldots \cup d_{n-1}$}.

4.3.2 The Correctness of Algorithm 4.3.1 (PQEP)

In the following discussion let G be the graph input to the planar graph constructor of Algorithm 4.3.1, let d_k be the set of leaf-nodes deleted from T_{k-1} during the k'th iteration of step 2 of Algorithm 4.3.4 and let e_k be the set of edges corresponding to the leaf-nodes of d_k. Also, let G_k be the portion of G that has been embedded after the k'th iteration of step 2 is complete; that is, $G_k = (V_k, E_{k'})$, where $E_{k'} = E_k - e_1 - e_2 - \ldots - e_k$.

Theorem 4.3.1 For any biconnected graph G with n vertices input to the planar graph constructor, the subgraph G*n* output by the constructor is planar.

Proof: The proof that G*n* is planar is performed by showing that each of the subgraphs G*k*, $1 \leq k \leq n$, formed by Algorithm 4.3.4 is planar.

(1) For k = 1, the subgraph G*1* is made up of the real vertex 1 which is planar. In addition, all the edges of F*2* are in the outer face of the embedding of G*1*.

(2) For $k \leq j$, assume that the subgraph G*k* is planar and that all the edges of F*k* lie in the outer face of the embedding of G*k*.

(3) For k = j+1, since G*j* is planar, all the edges of F*j* lie in the outer face of the embedding of G*j* and T*j* is the pq-tree representation of B*j* = G*j* U (W*j*, F*j*). After step 2 of Algorithm 4.3.3 is performed for k = j+1, the leaf-nodes in S*j+1* - d*j+1* are adjacent in the resulting pq-tree T*j'(m')*. Therefore, G*j+1* is planar and the edges of F*j+1* lie in the outer face of the embedding of G*j+1*.

By (1), (2), and (3) above, and the principle of induction, if follows that the subgraph G*k* is planar for $1 \leq k \leq n$. Thus G*n* is planar.

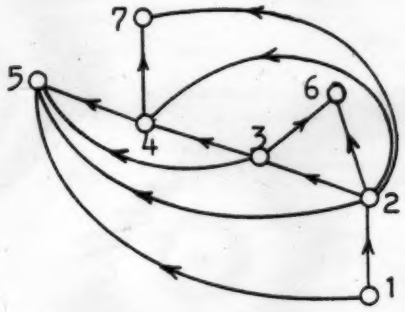
Theorem 4.3.2 Each subgraph H*j* of the partition {H*1*, H*2*, ..., Hq} constructed by Algorithm 4.3.1 (PQEP) is planar.

Proof: Each subgraph B*i'* constructed in step 2.2.1 of Algorithm 4.3.1 is planar by Theorem 4.3.1. Since any subgraph H*j* of the partition, $1 \leq j \leq q$, is the union of the planar subgraphs B*i'* of the biconnected components B*i* of H, $1 \leq i \leq r$, it follows that H*j* is planar.

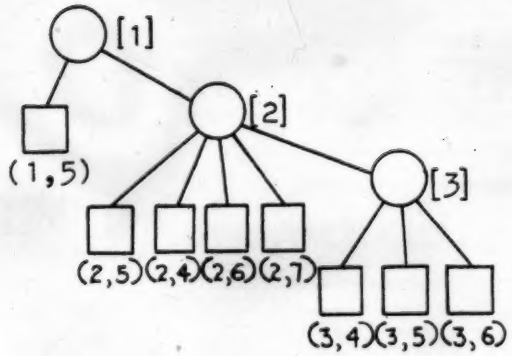
Unfortunately, there is no guarantee that, during step 2 of Algorithm 4.3.3, the subgraph G*k-1* of G = (V, E) and hence the graph with edge set E - e*1* - ... - e*k* still possesses an st-numbering of its vertices V. For this reason, after the first deletion of edges from G, it is possible that the remaining graph is planar when the modified embedding procedure declares it nonplanar. That is, the modified embedding procedure may only be detecting a possible cross-over rather than any unavoidable cross-over because of the lack of an st-numbering (see Figure 30). Thus the procedure may be deleting more edges from G than it is really necessary to delete.

4.3.3 Analysis of Time Complexity

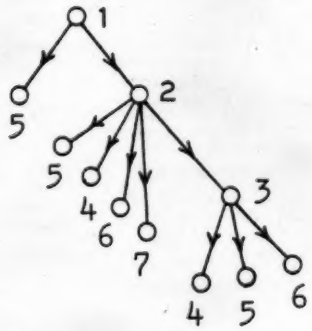
The planar graph constructor examines each edge of the input graph B*i* once and performs a constant number of operations for each edge. Thus, to obtain a planar subgraph H*j* of H, requires O(E(H)) operations or approximately O(e)



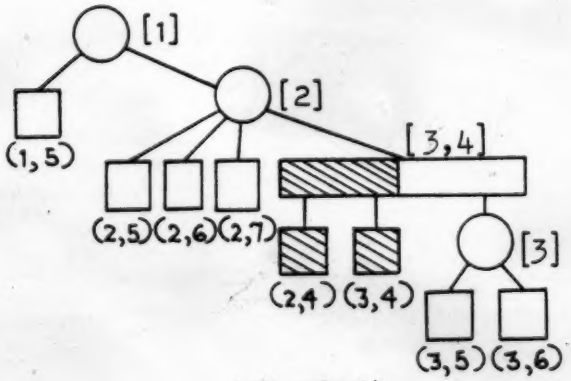
(a) The graph G



(b) T3

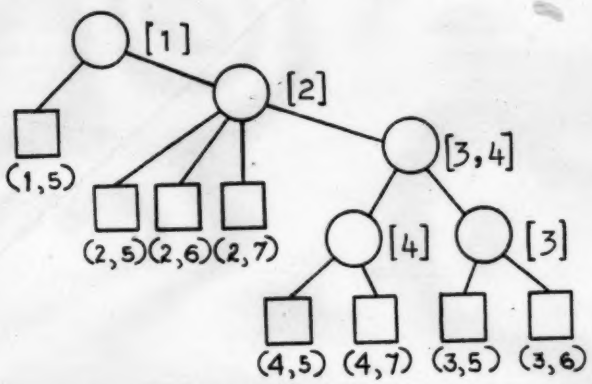


(c) B3

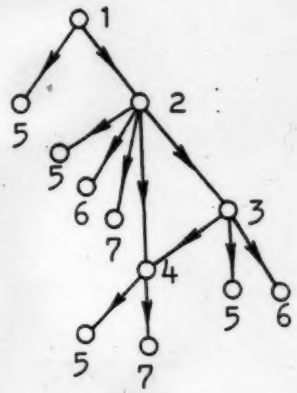


(d) T3(4)

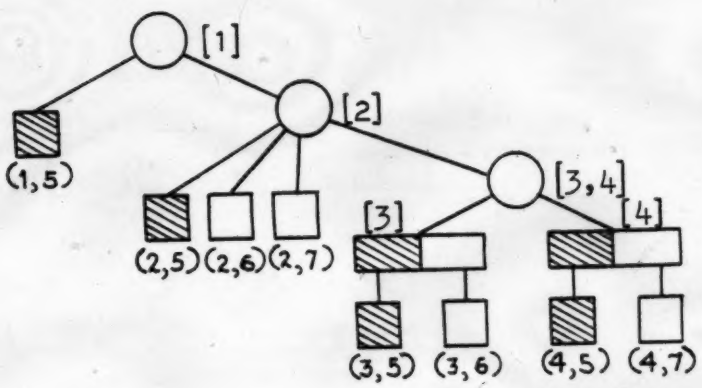
Figure 30. Embedding a Graph whose Vertices are not st-Numbered



(e) T4



(f) B4



(g) T4(6)

(h) The node [3,4] is visited next and cannot be matched to a template implying that G is nonplanar.

Figure 30. Embedding a Graph whose Vertices are not st-Numbered

operations, where e is the number of edges in the graph G. Since the order of the constructed planar partition is 0(n), the time complexity of the algorithm is 0(ne) (or at most $0(n^3)$ since $e < n^2$).

4.3.4 The PQEP Algorithm Summary

The pq-tree Edge Partitioning (PQEP) algorithm partitions the set of edges of a graph G producing a planar partition of G. Each subgraph in the partition is the union of the planar subgraphs Bi' of the biconnected components Bi of H, i = 1, 2, ..., r. Bi' is produced by a planar graph constructor which is a modified version of the pq-tree planarity algorithm from section 3.2. The constructor produces a planar graph by deleting edges from the input graph whenever the graph is declared nonplanar. Edges chosen for deletion are taken from the portion of the graph being embedded when nonplanarity is declared. If the graph is declared nonplanar when merging the virtual vertices k+1, then the edges chosen for deletion come from the set of edges with virtual vertex k+1. Edges to real vertices and edges to virtual vertices not yet visited are not considered for deletion.

An attempt was made to have the constructor delete the fewest number of edges required for the embedding process to continue. The combination of minimum edge deletion and local edge deletion was employed in an attempt to efficiently produce planar subgraphs of as large a size as possible.

Although the subgraphs constructed by the PQEP algorithm are planar, they are not always maximally planar for two reasons. First, as mentioned at the end of section 4.3.2, once edges are deleted from a biconnected graph, the graph may no longer be st-numbered. Therefore, it may be declared nonplanar when it is in fact planar. Secondly, it may not be possible to make the pertinent leaf-nodes of the retained group adjacent thus requiring more edge deletions. This may result in a greater number of total edge deletions than if a smaller group had been retained initially.

Chapter 5

Experimental Analysis of the STEP, PFEP, and PQEP Algorithms

The Spanning Tree Edge Partitioning (STEP) algorithm, Path Finding Edge Partitioning (PFEP) algorithm, and pq-tree Edge Partitioning (PQEP) algorithm are described in Chapter 4. Each one partitions the edges of a graph to produce a planar partition. This chapter compares these algorithms as heuristics for the minimal planar partition problem. Three comparisons are made. The first is based on the quality of the solution obtained as measured by how close the order of the constructed partition approximates the thickness of the graph. The second comparison is based on the resource requirements (execution time) of each. Finally, the characteristics of the constructed partitions are examined and compared to obtain some idea of each algorithm's usefulness as an aid in the design of circuit layouts.

5.1 The Experiments

The three algorithms were programmed in ALGOLW [Sit 72] and run on an AMDAHL V8 under the MTS Operating System.

Execution times were measured using CPU time. CPU time is measured in units of 1/60 seconds and is the time since the program started exection. The difference in CPU time before the construction of a partition and CPU time after the partition's formation is taken as the execution time.

The graphs used in the experiments are shown in Table I. A graph labelled G(n,p1,p2,...,pj) has n vertices and is the union of the j planar subgraphs Hp1, Hp2, ..., Hpj of the minimal planar partition for Kn described in section 2.2.1. For example, the graph G(30,2,4) has 30 vertices and is the union of the subgraphs H2 and H4 of the minimal planar partition for K30.

The graphs listed in Table I were chosen because the thickness, t(G), of each one is known. The complete graphs Kn, complete bipartite graphs K(m,p), and m-cubes Qm provide data on the behavior of the algorithms on regular graphs including regular dense graphs (a large number of edges per vertex) and regular sparse graphs. The graphs of the form G(n,p1,...,pj) provide data on the algorithms' behavior on irregular degree graphs, both sparse irregular (for smaller values of j) and dense irregular.

5.2 Performance Measure

The performance measure used to assess how well the STEP algorithm, the PFEP algorithm, and the PQEP algorithm perform as heuristic algorithms for the minimal planar problem is:

(1) For a graph G, with thickness t(G), let q be the number of subgraphs in the planar partition constructed by one of the algorithms. Then PM = q - t(G) is a measure of the performance of the algorithm.

Table I Test Graphs

Regular	Irregular
K9	G(30,1,5)
K11	G(30,2,4)
K16	G(30,1,3,5)
K17	G(30,2,3,4)
K30	G(30,1,2,4,5)
K40	G(30,1,2,3,4,5)
K60	
	G(42,3,6)
K(11,11)	G(42,2,3,5,7)
K(14,14)	G(42,1,2,3,4,5,6,7)
K(16,16)	
K(20,20)	G(300,1,9)
K(27,27)	G(300,10,31)
K(30,20)	
	G(600,20,55)
Q5	
Q9	

If PM is small, then the algorithm is considered to obtain a good approximation to the minimal planar partition for the graph.

5.3 Performance of the STEP, PFEP, and PQEP Algorithms

Table II shows the thickness, t(G), of each graph G used and the number of subgraphs, STEP(G), PQEP(G), and PFEP(G), in the constructed planar partitions. The blank entries in the table correspond to tests that were not run on the STEP algorithm because of the prohibitive amount of execution time those tests would have required. From the data in Table II it appears that the three algorithms

Table II The Number of Subgraphs t(G) in a Minimal Planar Partition of G and the Number of Subgraphs in the Constructed Planar Partitions

G	t(G)	STEP(G)	PQEP(G)	PFEP(G)
K9	3	3	3	3
K11	3	3	3	3
K16	3	5	5	4
K17	4	5	5	5
K30	6	8	8	8
K40	7		10	11
K60	11		15	16
K(11,11)	4	4	4	4
K(14,14)	4	5	5	5
K(16,16)	5	6	6	6
K(20,20)	6		7	7
K(27,27)	8		10	10
K(30,20)	7		9	9
Q5	2	2	2	2
Q9	3		4	4
G(30,1,5)	2	3	4	4
G(30,2,4)	2	4	4	4
G(30,1,3,5)	3	5	5	6
G(30,2,3,4)	3	5	5	5
G(30,1,2,4,5)	4	7	6	6
G(30,1,2,3,4,5)	5	8	8	8
G(42,3,6)	2	4	4	4
G(42,2,3,5,7)	4		6	7
G(42,1,2,3,4,5,6,7)	7		10	11
G(300,1,9)	2		5	6
G(300,10,31)	2		4	5
G(600,20,55)	2		5	5

produce approximately the same size partition for a graph. The constructed partitions are of order $q < \{e/n\}$ for any graph tested. The algorithms appear to perform best when determining a partition for a regular graph. In that case, the order of the partition is approximately $\{e/n\}/2$ and closer to the actual thickness of the graph.

For the complete graphs tested, the algorithms construct partitions of order

$$q \leq \{e/(2n-2)\} = \{n/4\} = [(n+3)/4].$$

Since the thickness of Kn is $[(n+7)/6]$ for $n \neq 9, 10$, the performance measure for the complete graphs is $PM = [(n+3)/4] - [(n+7)/6]$ or $PM \leq \{t(G)/2\}$. The algorithms' performance on the complete bipartite graphs and the m-cubes is even better with $PM \leq \{t(G)/4\}$. The algorithms have the most difficulty when constructing a planar partition for the sparse irregular graphs. The order of the constructed partition for these graphs is very close to $\{e/n\}$ while the actual thickness is much less than $\{e/n\}$. The number of edges in the irregular graphs G, listed in Table II, is $e = t(G)(3n-6)$. Therefore, $\{e/n\} = \{t(G)(3n-6)/n\} = 3t(G) - \{6t(G)/n\}$. Since $t(G) \leq [(n+7)/6]$, $\{6t(G)/n\} \leq \{(n+7)/n\} = 2$, for $n \geq 7$, it follows that $3t(G)-2 \leq \{e/n\} \leq 3t(G)-1$ for the irregular graphs. Thus the order of the constructed partitions for the sparse irregular graphs is within $2t(G)$ of the order of their minimal planar partition. For the denser irregular graphs, the performance measures for all three algorithms improve and approach the performance

measures for the complete graphs. For example, for G(42,1,2,3,4,5,6,7), PM is approximately {t(G)/2} for the three algorithms. Table III shows the performance measure, PM, of the algorithms for the regular graphs and irregular graphs used in the tests.

Although no algorithm performed markedly better than any of the others, the PQEP algorithm outperformed the PFEP algorithm on the larger complete graphs tested (K40, and K60) and on some of the irregular graphs (G(30,1,3,5), G(300,10,31)). The slight performance differences between the algorithms may be the result of the different way each one orders and traverses the vertices and edges of a graph. In addition, for each algorithm, the order in which the vertices and edges of a graph are processed appears to affect the performance of that algorithm performs. For example, for the PFEP algorithm, the constructed partition for G(30,1,3,5) contains six planar subgraphs while the partition for G(30,2,3,4) contains only five. For the PQEP algorithm, the constructed partition for G(300,1,9) is of order five while the constructed partition for G(300,10,31) is only of order four. The STEP algorithm exhibits the same behavior. For example, the partition it constructs for G(30,1,5) has three subgraphs and its partition for G(30,2,4) has four subgraphs. In each of these cases, the different order in which the vertices and edges of the graph are processed affects the size of the constructed partition. In general, the algorithms perform best on the regular

Table III Performance Measure PM of the STEP, PQEP, and PFEP Algorithms

Graphs	STEP	PQEP	PFEP
Regular Complete	{t(G)/2}	{t(G)/2}	{t(G)/2}
Regular not Complete	{t(G)/4}	{t(G)/4}	{t(G)/4}
Irregular Sparse	2t(G)	2t(G)	2t(G)

graphs tested, where the initial processing order of the vertices is of no consequence.

5.4 Resource Requirements

The resource requirement considered is the execution time required to obtain a planar partition for a graph. To partition a graph G = (V, E), where |V| = n and |E| = e, the time complexity, in the worst case, is 0(ne) for the PFEP and PQEP algorithms and $0(n^2e)$ for the STEP algorithm. Table IV shows the actual execution time required by the algorithms to obtain a partition for each test graph. The blank entries in the second column of Table IV correspond to tests not run on the STEP algorithm. As can be seen from the entries above the blank ones the execution time would have been prohibitively large for those tests. As such, those runs were deemed infeasible.

There are a number of observations that can be made regarding the execution times shown in Table IV. First, the

Table IV Execution Time in Seconds

G	STEP	PQEP	PFEP
K9	0.10	0.07	0.01
K11	0.22	0.10	0.03
K16	1.05	0.30	0.12
K17	1.35	0.35	0.13
K30	15.20	1.82	0.67
K40		4.23	1.52
K60		13.40	5.08
K(11,11)	1.98	0.28	0.10
K(14,14)	5.10	0.53	0.20
K(16,16)	8.60	0.75	0.30
K(20,20)		1.37	0.53
K(27,27)		3.10	1.23
K(30,20)		2.43	0.93
Q5	0.77	0.10	0.05
Q9		4.80	2.00
G(30,1,5)	3.43	0.33	0.12
G(30,2,4)	3.30	0.32	0.13
G(30,1,3,5)	8.03	0.63	0.27
G(30,2,3,4)	7.75	0.63	0.25
G(30,1,2,4,5)	13.90	1.03	0.43
G(30,1,2,3,4,5)	21.60	1.68	0.62
G(42,3,6)	6.40	0.45	0.20
G(42,2,3,5,7)		1.47	0.62
G(42,1,2,3,4,5,6,7)		4.42	1.70
G(300,1,9)		4.03	1.70
G(300,10,31)		3.47	1.63
G(600,20,55)		8.33	3.57

STEP algorithm requires the longest time to produce a partition, and the PFEP algorithm requires the shortest time. Second, the ratio of the execution times for the PQEP algorithm and the PFEP algorithm is constant as suggested by their time complexities (both O(ne)). The PQEP algorithm requires two to three times longer to obtain a partition than the PFEP algorithm. For example, the PQEP algorithm requires 2.8 times longer than the PFEP algorithm to produce a partition for K(11,11) and 2 times longer to produce a partition for Q5. The different sizes of the constructed partitions and the number of edges left to partition after each subgraph is constructed may be responsible for the observed fluctuation in the execution time ratio. That is, if the PQEP algorithm constructs a partition of smaller size than the PFEP algorithm, then the execution time ratio is closer to 2::1, as is the case for G(300,10,31), and if the partition size is the same but the PFEP algorithm has less edges to deal with after each subgraph is constructed, then the execution time ratio is closer to 3::1, as for G(30,1,5).

The third observation is that the execution time ratios of the STEP algorithm and the other two algorithms varies as the size (number of vertices and edges) of the graph varies, as suggested by their time complexities.

5.5 Characteristics of the Constructed Planar Partitions

If the algorithms are used as aids in the design of circuit layouts, then each subgraph in a constructed partition serves as a pattern for a layer (board) of the circuit being designed. In that case, the characteristics of each subgraph and of the partition as a whole are important. If a goal of the layout is to place as many wires as possible on one layer [Hig 73 and Lui 76], then a partitioning scheme that consistently achieves such optimal wire packing is more useful than one that does not. Constraints on wire density and wire length on a board may affect the usefulness of a partition as well. In addition, if there are any constraints on the number of boards on which a given component can appear [Rub 73], then a partition which cannot satisfy the constraints is not useful.

Following are some characteristics of the planar subgraphs and partitions constructed by the algorithms. Table V lists the maximum number of edges in a planar subgraph of the test graphs [Lui 76] and, for each algorithm, the number of edges placed in the largest constructed planar subgraph for each graph. The blank entries in the table correspond to tests that were not run on the STEP algorithm because of the prohibitive amount of execution time those tests would have required.

For the regular graphs shown in Table V, the largest planar subgraph of the constructed partitions is usually the

Table V Number of Edges in the Largest Planar Subgraph

MNE - the maximum number of edges in a planar subgraph of G.

G	MNE	STEP	PQEP	PFEP
K9	21	21	21	21
K11	27	27	27	27
K16	42	42	42	42
K17	45	45	45	45
K30	84	84	84	84
K40	114		114	114
K60	174		174	174
K(11,11)	40	40	40	40
K(14,14)	52	52	52	52
K(16,16)	60	60	60	60
K(20,20)	76		76	76
K(27,27)	104		104	104
K(30,20)	96		96	87
Q5	60	60	60	60
Q9	1020		892	1020
G(30,1,5)	84	76	64	74
G(30,2,4)	84	63	71	64
G(30,1,3,5)	84	80	76	76
G(30,2,3,4)	84	72	78	75
G(30,1,2,4,5)	84	81	80	83
G(30,1,2,3,4,5)	84	84	83	84
G(42,3,6)	120	85	96	90
G(42,2,3,5,7)	120		108	113
G(42,1,2,3,4,5,6,7)	120		119	120
G(300,1,9)	894		630	661
G(300,10,31)	894		679	715
G(600,20,55)	1794		1232	1444

same size as the maximum planar subgraph for that graph. This means that the algorithms have achieved to some degree the desired property of creating subgraphs containing as many edges as possible (the algorithms delete as few edges as possible). This is especially important for the PFEP and PQEP algorithms, where the maximality of the constructed subgraphs cannot be guaranteed (section 4.2.2 and section 4.3.2). A planar subgraph of maximum size is constructed for all regular graphs partitioned by the PFEP and STEP algorithms and for all complete graphs and complete bipartite graphs partitioned by the PQEP algorithm.

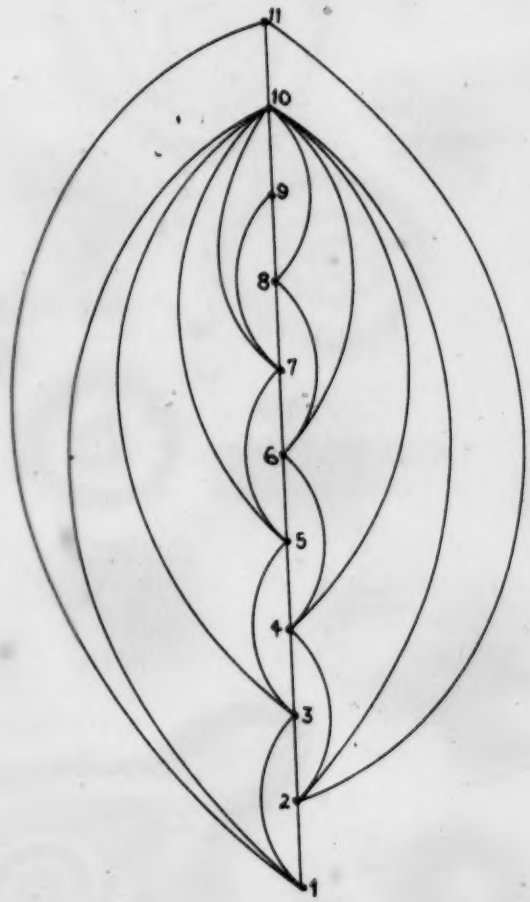
Also, for each algorithm, if the subgraphs in a constructed partition are produced in order $H1$, $H2$, ..., Hq, then $|E(H1)| \geq |E(H2)| \geq \ldots \geq |E(Hq)|$. Again, it appears that the algorithms are deleting as few edges as possible from a nonplanar graph.

Each subgraph constructed by the STEP algorithm is maximally planar with respect to the edges that remain to be partitioned when the subgraph is constructed. The subgraphs constructed by the other two algorithms cannot be guaranteed to be maximally planar. Nevertheless, for the PFEP algorithm, it appears that the subgraphs are maximally planar or nearly so. There are two justification for this claim. First, as Table V shows, for regular graphs, the largest planar subgraph is the maximum size possible. Secondly, when a subgraph constructed by the PFEP algorithm was input to a maximization procedure, at most three edges

were added to the subgraph implying that the subgraph is nearly maximally planar.

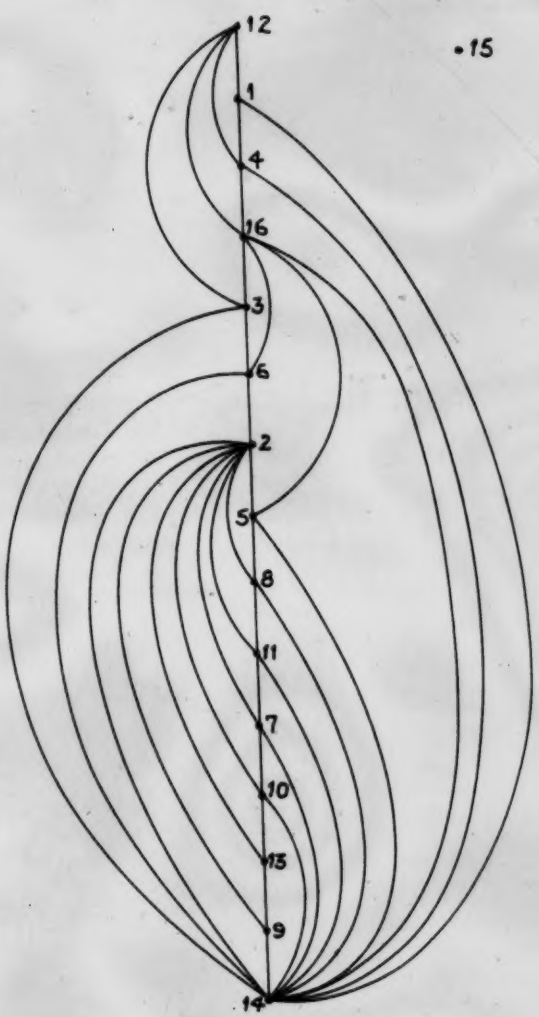
When the PFEP algorithm was run with a maximization procedure one observation made was that the size of the constructed partition is the same as the size of the partition constructed when the PFEP algorithm is run alone. This implies that, regardless of maximization, the sizes of the partitions constructed by the algorithm are the smallest possible for this type of algorithm.

Figure 31(a), (b), and (c) illustrate the subgraphs produced by the PFEP algorithm. Each subgraph B*i* input to the PFEP algorithm's planar graph constructor has all of the edges of one of its vertices placed in B*i'*. This occurs for the vertex numbered 1 in the directed version of B*i*. Thus, for graphs with vertices of large regular degree k, the subgraphs constructed by the PFEP algorithm are characterized by at least one vertex of degree k. In addition, when this occurs, the remaining vertices are of small degree (degree 3, 4, or 5) as illustrated by Figure 31(a) and (b). Each subgraph B*i'* produced by the PQEP algorithm's planar graph constructor has two vertices with all of their incident edges from B*i*. If B*i* has n vertices, then the vertex with st-number n has all its incident edges placed in B*i'* and very often the vertex with st-number 2 has all its incident edges placed in B*i'* as well. This means that, for large regular or nearly regular graphs, the subgraphs produced by the PQEP algorithm, have at least two



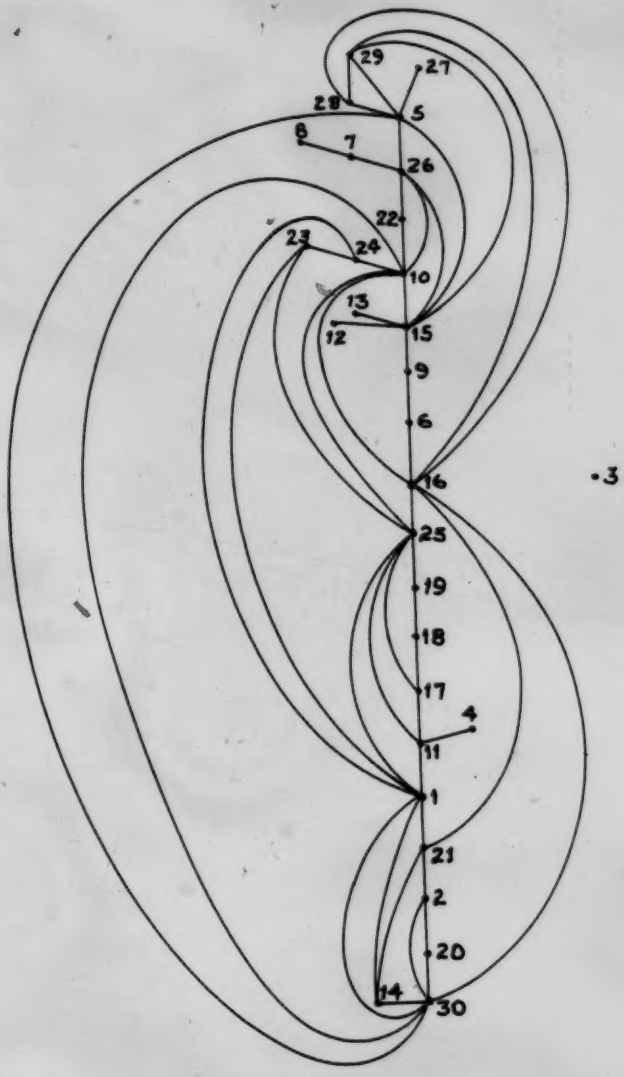
(a) H1 of K11

Figure 31. Subgraphs Constructed by PFEP



(b) H2 of K16

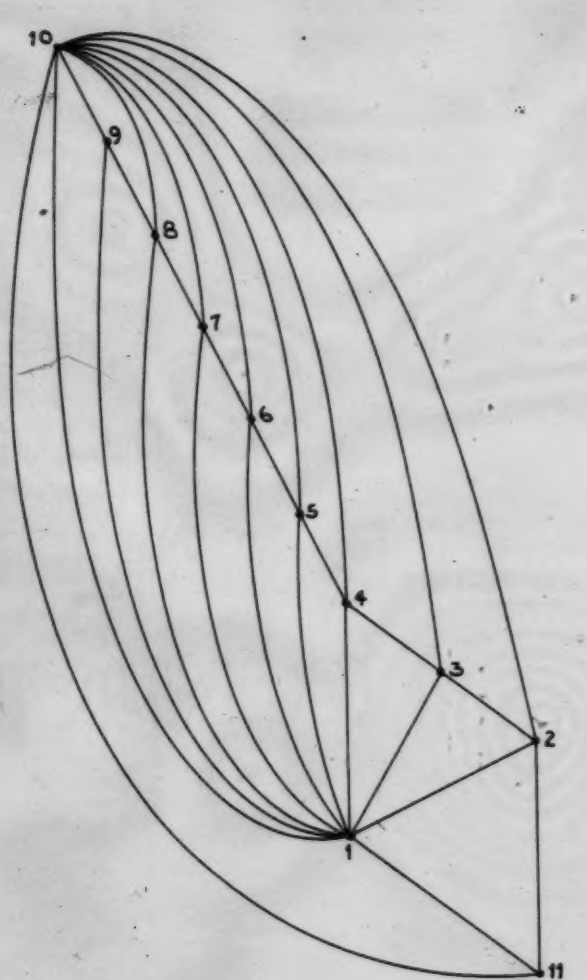
Figure 31. Subgraphs Constructed by PFEP



(c) H2 of G(30,1,5)

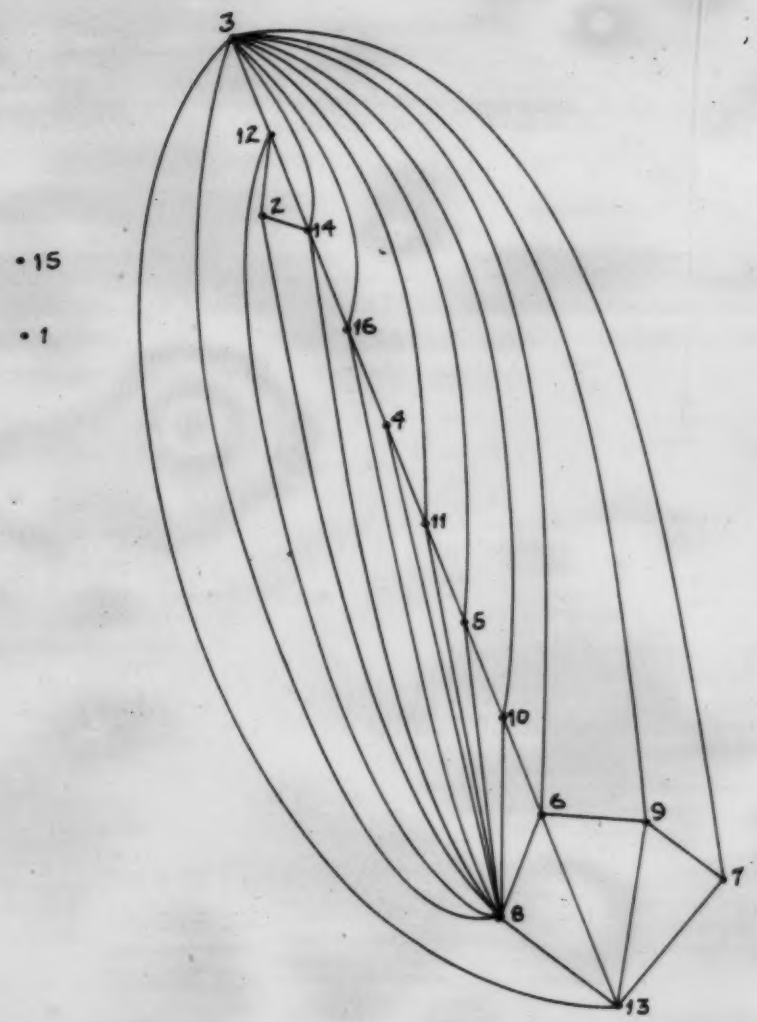
Figure 31. Subgraphs Constructed by PFEP

vertices of large degree and the remaining vertices of small degree as shown in Figure 32(a), (b), and (c). The STEP algorithm also tends to construct subgraphs that have at least one vertex of higher degree than the rest of the vertices as illustrated by Figure 33(a), (b), and (c). This tendency is a result of the order in which the edges remaining to be partitioned (the edges in S) are processed in step 2.2.3.1 of Algorithm 4.2.1. A change in the processing order of the edges may result in the construction of planar subgraphs of almost regular degree. Subgraphs having one or more vertices of large degree may not be very useful as patterns for circuit layouts because of such problems as wire density and wire length. On the other hand, because at least one vertex has edges in a subgraph H_j and does not have any edges in a subgraph H_i, $1 \leq j < i \leq q$, any constraints on the number of boards on which a module can appear may be satisfied by the partitions.



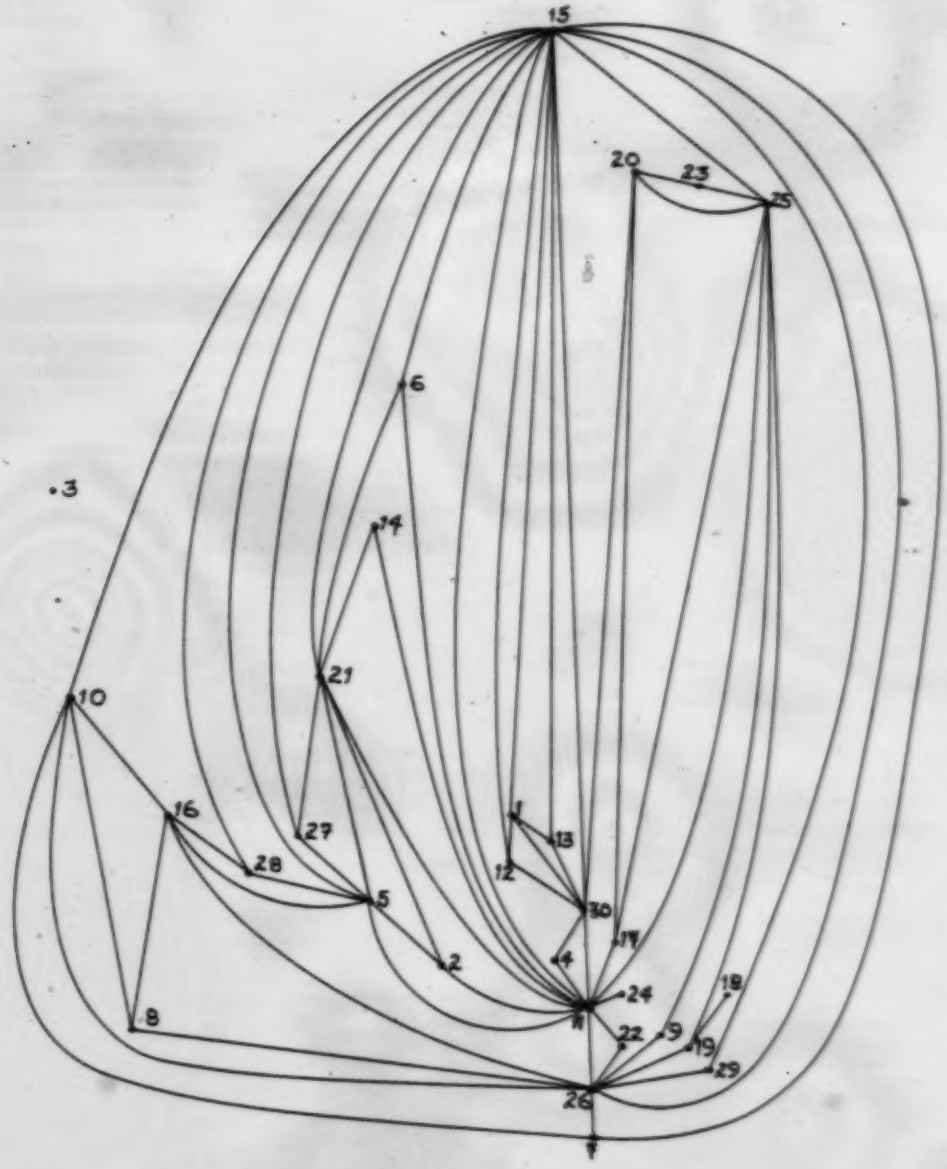
(a) H1 of K11

Figure 32. Subgraphs Constructed by PQEP



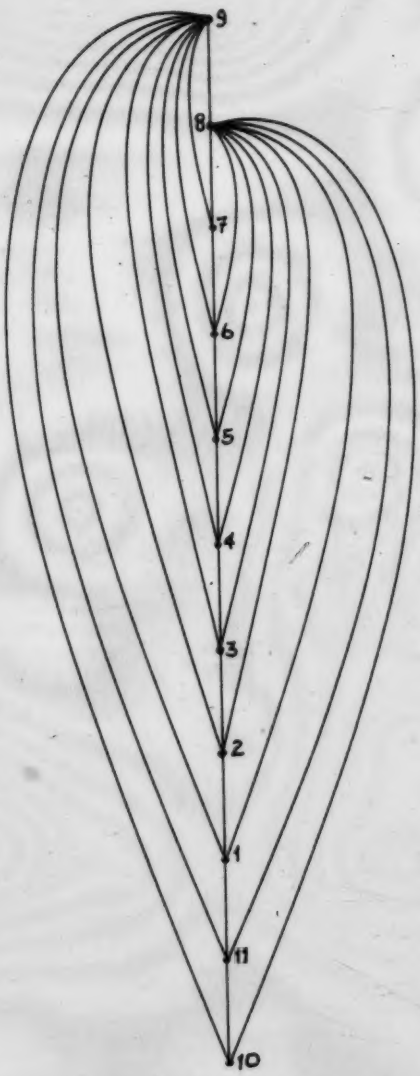
(b) H2 of K16

Figure 32. Subgraphs Constructed by PQEP



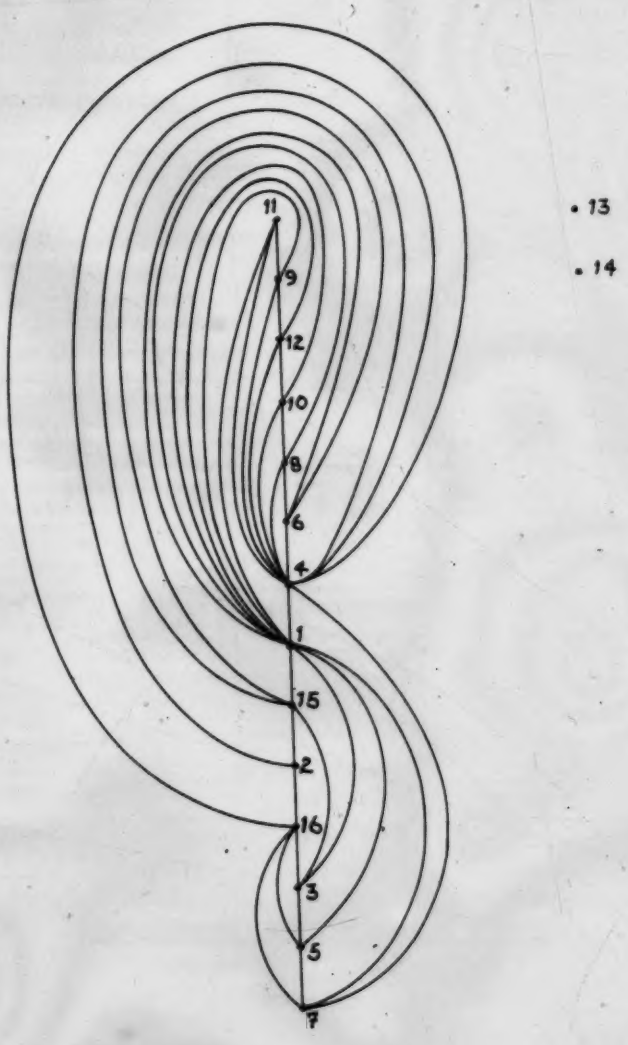
(c) H2 of G(30,1,5)

Figure 32. Subgraphs Constructed by PQEP



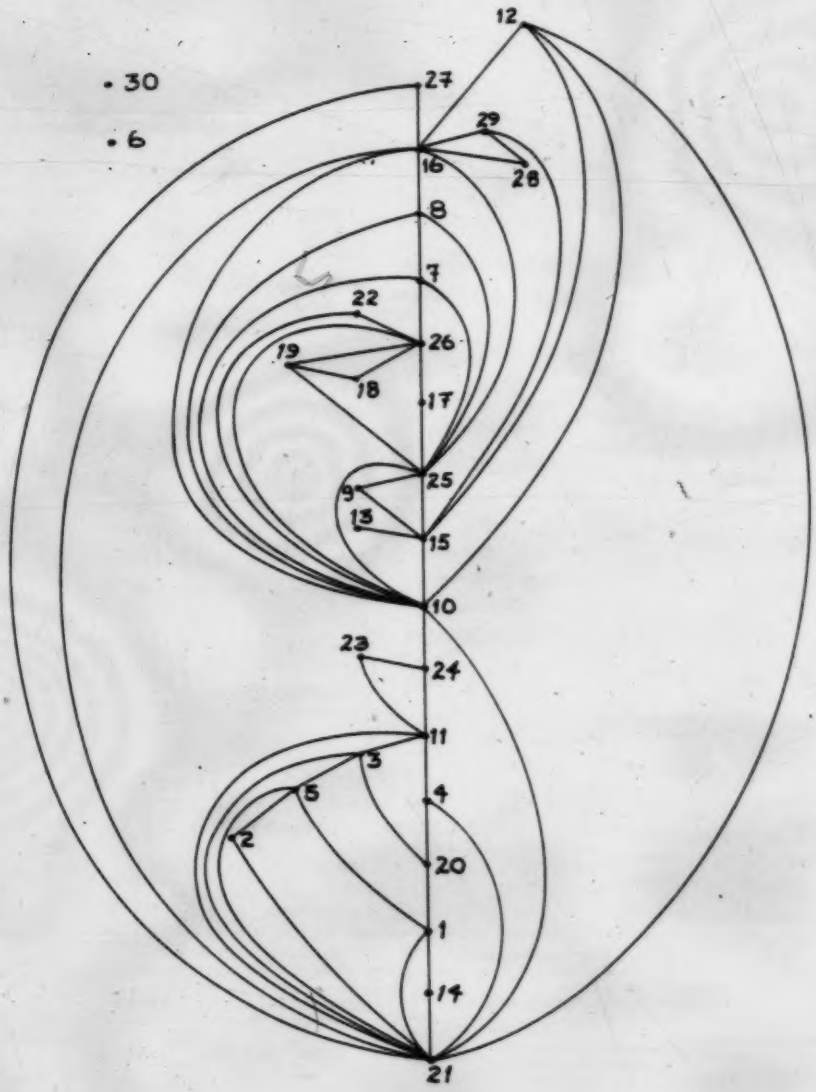
(a) H1 of K11

Figure 33. Subgraphs Constructed by STEP



(b) H2 of K16

Figure 33. Subgraphs Constructed by STEP



(c) H2 of G(30,1,5)

Figure 33. Subgraphs Constructed by STEP

Chapter 6

Summary and Conclusions

6.1 Summary

The survey in Chapter 2 examines the thickness results and minimal planar partitions for complete graphs and complete bipartite graphs, thickness results for m-cubes, and minimum size planar partitions with degree constrained subgraphs for the complete graphs and the complete bipartite graphs.

Two linear time planarity algorithms, a path finding algorithm and a pq-tree algorithm, are examined in Chapter 3 and a logic error that appeared in an earlier description of the path finding algorithm is corrected.

Chapter 4 contains a description of three new planar edge partitioning algorithms. The algorithms partition the edges of a graph G = (V, E) into nonempty, planar, disjoint subsets. A common feature of each algorithm is that each one produces a single planar partition for a graph and makes no attempt to improve on the partition through iteration.

The first new algorithm described, the Spanning Tree Edge Partitioning (STEP) algorithm differs from the other two in its approach to constructing each subgraph of the partition for a graph G. It starts with a spanning tree (spanning forest) of the unpartitioned graph H as an initial

approximation to a maximally planar subgraph of H. Next, for each edge in H but not in the tree, the STEP algorithm checks if the edge can be added to the subgraph under construction (maintaining planarity) in an attempt to improve on the first approximation (increase the number of edges in the subgraph). After all remaining edges of H are tested, the constructed subgraph is accepted as a member of the planar partition.

The other two new algorithms described, the Path Finding Edge Partitioning (PFEP) algorithm and the pq-tree Edge Partitioning (PQEP) algorithm, construct a planar partition for G using modified versions of planarity algorithms. Each one constructs a planar subgraph by repeatedly deleting edges from the unpartitioned graph until the resulting subgraph is planar. The PFEP algorithm uses a modified version of the path finding planarity algorithm described in section 3.1 and the PQEP algorithm uses a modified version of the pq-tree algorithm described in section 3.2. The embedding procedure of each planarity algorithm was modified so that it would return a planar subgraph of the input graph. Modifications were introduced that result in the deletion of a small number of edges from the input graph to efficiently produce a planar subgraph.

Proofs that the three algorithms produce planar subgraphs are given in Chapter 4. The STEP algorithm produces maximally planar subgraphs while the subgraphs produced by the PFEP algorithm and the PQEP algorithm are

not guaranteed to be maximally planar. Upper bounds on the time complexities of the algorithms are obtained: $O(n^2e)$ for the STEP algorithm, and $O(ne)$ for the PFEP and the PQEP algorithms.

Chapter 5 examines the value of the new algorithms as heuristic algorithms for the minimal planar partition problem. The implemented algorithms were run on graphs for which the order of the minimal planar partition (thickness) is known.

The algorithms obtain good approximations for all graphs tested. They produce the best approximations (the performance measure PM is $t(G)/4$) for the regular sparse graphs and the poorest approximations ($PM = 2t(G)$) for the irregular sparse graphs tested.

The three algorithms are also compared on the basis of their execution times and the characteristics of their constructed partitions. The ratio of execution times of the PQEP algorithm and the PFEP algorithm is constant with the PFEP algorithm two to three times faster than the PQEP algorithm. The ratio of execution times of the STEP algorithm and the PFEP algorithm varies, as expected, as the size of the input graph varies. The PFEP algorithm is $O(n)$ times faster than the STEP algorithm.

One characteristic of the constructed partitions for the complete graphs and the complete bipartite graphs is that the largest subgraph is of maximum planar size. Also, the STEP algorithm and the PFEP algorithm construct a

subgraph of maximum planar size for the m-cubes. The subgraphs constructed by the STEP algorithm are always maximally planar and those constructed by the PFEP and PQEP algorithms are maximally planar or nearly so. The STEP algorithm has a tendency to construct subgraphs with one or two vertices of larger degree than the rest, the PFEP algorithm produces subgraphs with at least one vertex of larger degree, and the PQEP algorithm produces subgraphs with at least two vertices of larger degree.

6.2 Conclusions

The first goal of this thesis, that of designing an efficient algorithm that produces a good approximation to a minimal planar partition for a graph, was achieved. Three efficient heuristic algorithms were designed.

The PFEP algorithm is the best algorithm of the three because it requires less time than the other two to produce planar partitions of the same order. The STEP algorithm requires much more execution time (0(n) more time) than the PFEP algorithm and this is the major drawback to its use.

It may be more desirable to use the PQEP algorithm when partitioning large complete graphs or irregular graphs since the algorithm produces partitions with one less subgraph than the PFEP algorithm produces for those test graphs.

The second goal, the desired balance between the complexity of an algorithm and the quality of the

approximation it produces, has also been achieved. Any consistent decrease in the sizes of the constructed (improvement in the approximation) partitions would probably require one or both of the following: an iteration scheme to improve on the initial partition, or, in the planar graph constructors of the PFEP and PQEP algorithms, the introduction of lookahead or backtracking to make a more informed decision about which edges to delete.

It may be feasible to use the algorithms in the design of circuit layouts. They produce good approximations to minimal planar partitions and the PFEP and PQEP algorithms do so efficiently. If a goal of the layout is to have each board contain as many wires as possible, then these algorithms may be useful in placing the wires on the boards. The PFEP and the STEP algorithms may be more useful than the PQEP algorithm as they consistently produce larger subgraphs. Because the algorithms produce subgraphs with at least one vertex of large degree (and the remaining vertices of small degree), the density of wires around the large degree vertices may become too large for the circuit. The PFEP algorithm is the most useful in this case because each constructed subgraph has only one vertex of large degree. The other two algorithms produce subgraphs with two or more vertices of large degree. However, since the PQEP algorithm partitions a graph, so that at least two vertices in one subgraph do not have edges in any subgraph produced later on, the algorithm may be useful in circuit layouts where

there are constraints on the number of boards on which a module may appear.

One major attribute of the STEP algorithm is its flexibility in producing subgraphs. Because edges are added to a subgraph one at a time, any violation of wire density (degree) constraints or wire length constraints could be checked before an edge is placed in the subgraph. With the PFEP and PQEP algorithms, constraint violation checking must occur after the subgraph is produced. Thus for constraints of this nature the PFEP and PQEP algorithms are less effective in designing a good layout for a circuit on a small number of boards.

6.3 Further Research

There are questions to be answered concerning the minimal planar partition problem for an arbitrary graph:

1. Is the problem of determining a minimal planar partition for an arbitrary graph NP-complete?
2. If the problem is not NP-complete, then what is a method for solving the problem in polynomial time?

Some questions that remain concerning Fisher and Wing's algorithm [Fis 66], outlined in Chapter 4, and the three new algorithms include:

1. How well does Fisher and Wing's algorithm perform and how does it compare with the STEP, PFEP, and PQEP algorithms?

2. Does the ordering of the vertices (by vertex degree) affect the size of the partitions produced by the algorithms and, if so, does the optimal ordering vary with the class of graph being partitioned (as occurs with graph coloring algorithms)?
3. Do the PFEP and PQEP algorithms produce subgraphs that are far from maximal size for some graphs?
4. Is it possible to use the edge partitioning algorithms proposed in this thesis to aid in the design of circuit layouts? How useful are they if there exist constraints on wire density on a board, wire routing and length, the placement of modules, and the number of boards on which each module is represented?
5. The tests performed on the proposed algorithms are not complete and more testing is required to determine their performance measures with greater accuracy. It may be the case that, for irregular graphs, the limit of 2t(G) for the performance measures is too small.

References

[Ale 76] Alekseev, V. B., and Goncakov, V. S. "Thickness of An Arbitrary Complete Graph." **Math USSR SBORNIK**, 30(2):187-202, 1976.

[Aus 61] Auslander, L., Parter, S. V. "On Imbedding Graphs in the Sphere." **Journal of Mathematics and Mechanics**, 10(3):517-523, 1961.

[Bat 62] Battle, J., Harary, F., and Kodama, Y. "Every Planar Graph with Nine Points has a Non-planar Complement." **Bulletin of the American Mathematical Society**, 68:569-571, 1962.

[Bei1 67] Beineke, L. W. "The Decomposition of Complete Graphs into Planar Subgraphs." **Graph Theory and Theorectical Physics**. Edited by F. Harary. New York: Academic Press, 1967, pp 139-153.

[Bei2 67] --------. "Complete Bipartite Graphs: Decomposition into Planar Subgraphs." **A Seminar in Graph Theory**. Edited by F. Harary. New York: Holt, Reinhart, and Winston, 1967, pp 42-53.

[Bei 65] --------., and Harary, F. "The Thickness of the Complete Graph." **Canadian Journal of Mathematics**, 17:850-859, 1965.

[Bei 64] --------., Harary, F., and Moon, J. W. "On the Thickness of the Complete Bipartite Graph." **Proceedings of the Cambridge Philosophical Society**, 60:1-5, 1964.

[Bon 76] Bondy, J. A., and Murty, U. S. R. **Graph Theory with Applications**. London: The MacMillan Press Ltd., 1976, 263 pages.

[Boo 76] Booth, K. S., and Lueker, G. S. "Testing for Consecutive Ones Property, Interval Graphs, and Graph Planarity Using Pq-tree Algorithms." **Journal of Computer and System Sciences**, 13(3):335-379, 1976.

[Bos 77] Bose, N. K., and Prabhu, K. A. "Thickness of Graphs with Degree Constrained Vertices." **IEEE Transactions on Circuits and Systems**, CSA-24(4):184-190, 1977.

[Deo 76] Deo, N. "Note on Hopcroft and Tarjan Planarity Algorithm." **Journal of the ACM**, 23:74-75, 1976.

[Eve 79] Even, S. **Graph Algorithms**. Maryland: Computer Science Press, Inc., 1979, 249 pages.

[Eve 76] --------., and Tarjan, R. E. "Computing an st-Numbering." **Theoretical Computer Science**, 2:339-344, 1976.

[Fer 70] Ferrari, D., and Mezzalira, L. "A Computer-Aided Approach to Integrated Circuit Layout Design." **Computer Aided Design**, 2:19-23, 1970.

[Fis 66] Fisher, G. J., and Wing, O. "Computer Recognition and Extraction of Planar Graphs from the Incidence Matrix." **IEEE Transactions on Circuit Theory**, CT-13:154-163, 1966.

[Gar 79] Garey, M. R., and Johnson, D. S. **Computers and Intractability A Guide to the Theory of NP-Completeness**. San Francisco: W. H. Freeman and Company, 1979, 340 pages.

[Han 75] Hanan, M. "Layout, Interconnection, and Placement." **Networks**, 5:85-88, 1975.

[Har 69] Harary, F. **Graph Theory**. Massachusetts: Addison-Wesley Publishing Company, 1969, 274 pages.

[Hig 73] Hightower, D. W. "The Interconnection Problem - A Tutorial." **Proceedings of the Tenth Annual Design Automation Workshop**, Portland, Oregon, pp 1-21, June 1973.

[Hob 69] Hobbs, A. M. "A Survey of Thickness." **Recent Progress in Combinatorics**. Edited by W. T. Tutte. New York: Academic Press, Inc., 1969, pp 255-264.

[Hop 74] Hopcroft, J. E., and Tarjan, R. E. "Efficient Planarity Testing." **Journal of the ACM**, 21:549-568, 1974.

[Lem 66] Lempel, A., Even, S., and Cederbaum, I. "An Algorithm for Planarity Testing of Graphs." **Theory of Graphs: International Symposium Rome 1966.** Edited by P. Rosenstiehl. New York: Gordon and Breach, 1967, pp 215-232.

[Lin 75] Lin, S., "Heuristic Programming as an Aid to Network Design." **Networks**, 5:33-43, 1975.

[Lui 76] Lui, P. "Analysis of Nonplanar Graphs." Ph D dissertation, Stevens Institute of Technology, 1976, 108 pages.

[Rei 77] Reingold, E. M., Nievergelt, J., and Deo, N. **Combinatorial Algorithms: Theory and Practice.** New Jersey: Prentice-Hall Inc., 1977, 433 pages.

[Rub 73] Rubin, F. "Assigning Wires to Layers of a Printed Circuit Board." **Proceedings of the Tenth Annual Design Automation Workshop**, Portland, Oregon, pp 22-32, June 1973.

[Sit 72] Sites, R. L. **ALGOL W Reference Manual** STAN-CS-71-230, Computer Science Department, Stanford University, February 1972, 141 pages.

[Tar 71] Tarjan, R. E. "An Efficient Planarity Algorithm." Ph D dissertation, Stanford University, November 1971, 154 pages.

[Tut1 63] Tutte, W. T. "The Thickness of a Graph." **Indag. Math.**, 25:567-577, 1963.

[Tut2 63] --------. "On the Non-biplanar Character of the Complete 9-Graph." **Canadian Mathematical Bulletin**, 6(3):319-330, 1963.

[Van 75] Van Cleemput, W. "Mathematical Models and Algorithms for the Circuit Layout Problem." Ph D dissertation, University of Waterloo, 1975.

[Vas 76] Vasak, J. M. "The Thickness of the Complete Graph." Ph D dissertation, University of Illinois, 1976, 70 pages.